МИНОБРНАУКИ РОССИИ

Передовые инженерные школы

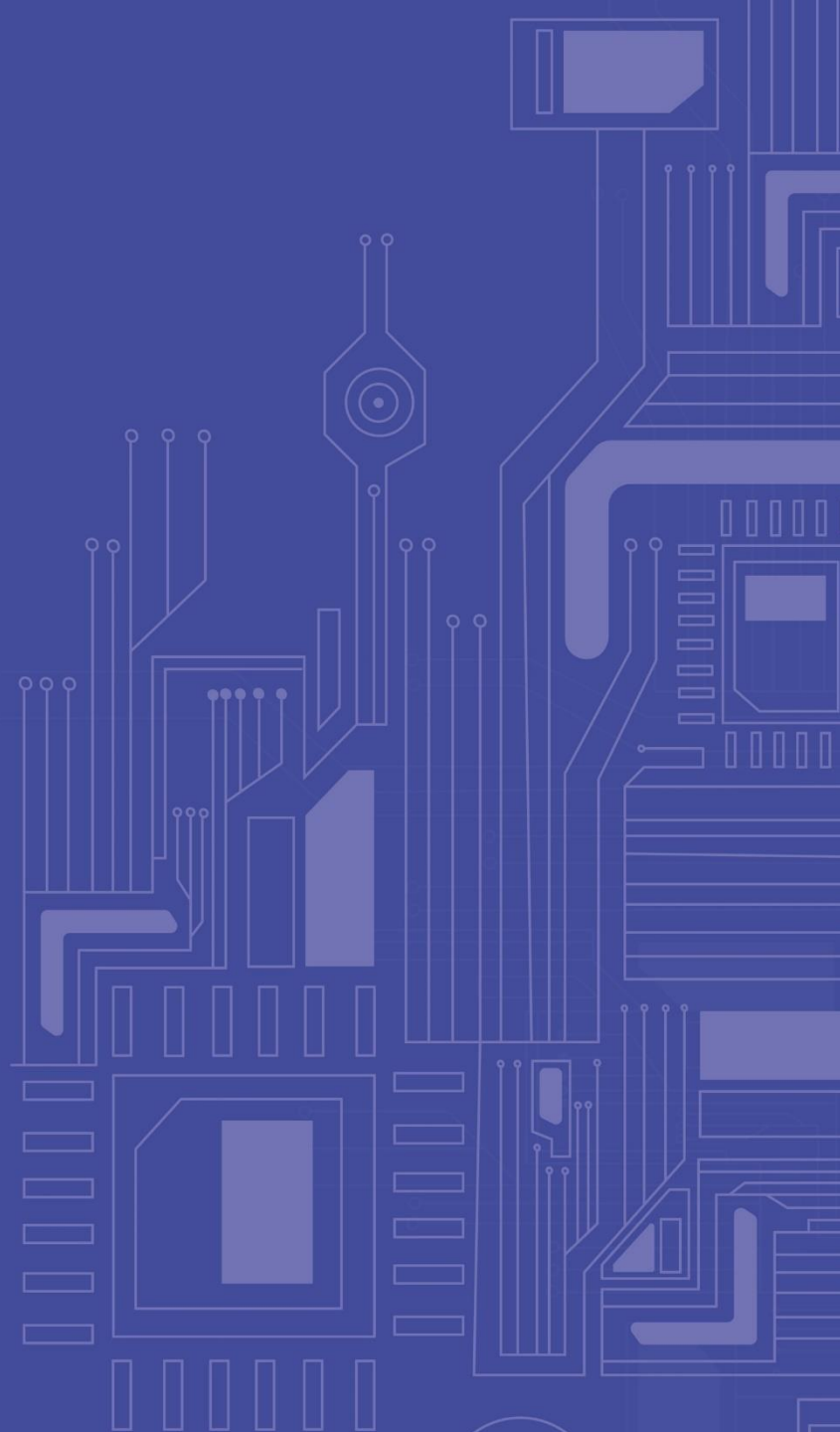
# СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

***Лекция 12***

# ПЛАН ЛЕКЦИИ

- Порождаемые и соединенные таблицы
- Аналитические запросы
- Базовые средства манипулирования данными

# ПОРОЖДАЕМЫЕ И СОЕДИЕННЫЕ ТАБЛИЦЫ



# РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ ЗАПРОСОВ

Использование ссылок на порождаемые таблицы расширяет возможности формулировки запросов.

**ПРИМЕР**

Найти номера отделов и имена руководителей отделов, которые числятся в тех же отделах, которыми руководят, и получают зарплату, размер которой является максимальным для служащих данного отдела.

```
SELECT MNG.DEPT_NO, MNG.MNG_NAME
FROM (SELECT DEPT.DEPT_NO, EMP.DEPT_NO, EMP_NAME, EMP_SAL
   FROM DEPT, EMP
   WHERE DEPT.DEPT_MNG = EMP.EMP_NO)
   AS MNG (DEPT_NO_1, DEPT_NO_2, MNG_NAME, MNG_SAL)
WHERE DEPT_NO_1 = DEPT_NO_2
   AND MNG_SAL = (SELECT MAX (EMP_SAL)
     FROM EMP
     WHERE EMP.DEPT_NO = DEPT_NO_1);
```

<=>

```
SELECT DEPT.DEPT_NO, EMP.EMP_NAME
FROM DEPT, EMP
   WHERE DEPT.DEPT_MNG = EMP.EMP_NO
     AND DEPT.DEPT_NO = EMP.DEPT_NO
     AND EMP.EMP_SAL = (SELECT MAX(EMP_SAL)
       FROM EMP
       WHERE EMP.DEPT_NO = DEPT.DEPT_NO);
```

В этом запросе порождаемая таблица MNG содержит по одной строке для каждого служащего, являющегося руководителем отдела. Первый столбец этой таблицы – DEPT_NO_1 – содержит номер отдела, которым руководит данный служащий. В столбце DEPT_NO_1 хранятся номера отделов, в которых числятся руководители отделов, а в столбцах EMP_NAME и EMP_SAL содержатся имя служащего-руководителя отдела и размер его заработной платы соответственно.

# КОГДА БЕЗ ПОРОЖДАЕМЫХ ТАБЛИЦ НЕ ОБОЙТИСЬ (1/2)

**ПРИМЕР** Найти общее число служащих и максимальный размер зарплаты в отделах с одинаковым максимальным размером зарплаты.

```
SELECT SUM (TOTAL_EMP), MAX_SAL
FROM (SELECT MAX (EMP_SAL), COUNT (*)
   FROM EMP
   WHERE DEPT_NO IS NOT NULL
   GROUP BY DEPT_NO ) AS DEPT_MAX_SAL (MAX_SAL, TOTAL_EMP)
GROUP BY MAX_SAL;
```

<=>

```
WITH DEPT_MAX_SAL (MAX_SAL, TOTAL_EMP) AS
   (SELECT MAX (EMP_SAL), COUNT (*)
     FROM EMP
     WHERE DEPT_NO IS NOT NULL
     GROUP BY DEPT_NO)
SELECT SUM (TOTAL_EMP), MAX_SAL
FROM DEPT_MAX_SAL
GROUP BY MAX_SAL;
```

Здесь мы не можем обойтись «одноуровневой» конструкцией запроса, поскольку требуется двойная группировка, причем вторая группировка должна быть получена в соответствии с результатами первой.

В этом случае выражение запросов, содержащееся в разделе FROM, можно перенести в раздел WITH

# КОГДА БЕЗ ПОРОЖДАЕМЫХ ТАБЛИЦ НЕ ОБОЙТИСЬ (2/2)

**ПРИМЕР** Найти число проектов, дату их завершения и средний размер зарплаты служащих, участвующих в проекте, для проектов с одной и той же датой завершения и одним и тем же средним размером зарплаты служащих, участвующих в проекте.

```
SELECT COUNT (*), PRO_EDATE, AVG_SAL
FROM (SELECT PRO_EDATE, AVG (EMP_SAL)
   FROM (SELECT PRO_SDATE + PRO_DURAT, PRO_NO
     FROM PRO) AS PRO1 (PRO_EDATE, PRO_NO), EMP
   WHERE PRO1.PRO_NO = EMP.PRO_NO
   GROUP BY PRO1.PRO_NO ) AS PRO_AVG_SAL (PRO_EDATE, AVG_SAL)
GROUP BY PRO_EDATE, AVG_SAL;
```

<=>

Для упрощения вида формулировки это выражение разумно вынести в раздел WITH

```
WITH PRO1 (PRO_EDATE, PRO_NO) AS
   (SELECT PRO_SDATE + PRO_DURAT, PRO_NO
     FROM PRO)
SELECT COUNT (*), PRO_EDATE, AVG_SAL
FROM (SELECT PRO_EDATE, AVG (EMP_SAL)
   FROM PRO1, EMP
   WHERE PRO1.PRO_NO = EMP.PRO_NO
   GROUP BY PRO1.PRO_NO) AS PRO_AVG_SAL (PRO_EDATE, AVG_SAL)
GROUP BY PRO_EDATE, AVG_SAL;
```

Выражение запросов на третьей и четвертой строках примера необходимо только по той причине, что нам требуется группировка по дате окончания проектов, соответствующий столбец в таблице PRO отсутствует, а в списке группировки можно использовать только имена столбцов.

# СОЕДИНЕННЫЕ ТАБЛИЦЫ (1/4)

В примерах присутствовало много запросов с соединениями двух или более таблиц. Условия соединения задавались предикатами сравнения столбцов таблиц, специфицированных в разделе FROM, и входили в состав логических выражений раздела WHERE (или, реже, раздела HAVING). Поскольку на практике требуются разные виды соединений, в стандарте SQL/92 появилась альтернативная возможность спецификации соединений – соединенная таблица (joined table). Соответствующая конструкция может использоваться в разделе FROM выражения запросов и фактически позволяет строить выражения соединений таблиц.

Синтаксические правила построения таких выражений выглядят следующим образом:

```
joined_table ::= cross_join
    | qualified_join
    | natural_join
    | union_join
cross_join ::= table_reference CROSS JOIN table_primary
qualified_join ::= table_reference [ join_type ] JOIN
   table_primary join_specification
natural_join ::= table_reference NATURAL [ join_type ]
   JOIN table_primary
union_join ::= table_reference UNION JOIN table_primary
join_type ::= INNER | { LEFT | RIGHT | FULL } [ OUTER ]
join_specification ::= ON conditional_expression
    | USING (column_comma_list)
```

# СОЕДИНЕННЫЕ ТАБЛИЦЫ

## Формальные определения

Пусть требуется выполнить некоторую операцию соединения над таблицами table1 и table2. Тогда:

Обозначим через CP результат выполнения запроса

SELECT * FROM table1, table2

- Если задается операция JOIN (или NATURAL JOIN) без явного указания типа соединения (join_type), то по умолчанию имеется в виду INNER JOIN (или NATURAL INNER JOIN).

- Если в спецификации соединения (join_specification) указано ключевое слово ON, то все ссылки на столбцы, встречающиеся в условном выражении (conditional_expression), должны указывать на столбцы таблиц table1 и table2 или на столбцы таблиц внешнего запроса. Если в этом условном выражении присутствует вызов агрегатной функции, то соединенная таблица может фигурировать только в подзапросах, используемых в разделах HAVING или SELECT внешнего запроса, и ссылка на столбец в вызове функции должна указывать на столбец таблицы внешнего запроса.

- Для прямых соединений (CROSS JOIN) и всех других видов соединения, включающих раздел ON, заголовок результата операции совпадает с заголовком таблицы CP.

# СОЕДИНЕННЫЕ ТАБЛИЦЫ: ЗАГОЛОВОК РЕЗУЛЬТАТА

Если в спецификации вида соединения присутствуют ключевые слова NATURAL или USING, то заголовок результата операции определяется следующим образом:

- если в спецификации вида соединения присутствует ключевое слово NATURAL, то будем называть *соответствующими столбцами соединения* (corresponding join column) все столбцы таблиц table1 и table2, которые имеют в заголовках этих таблиц одинаковые имена. Если в спецификации вида соединения присутствует ключевое слово USING, то будем называть соответствующими столбцами соединения (corresponding join column) все столбцы таблиц table1 и table2, имена которых входят в список имен столбцов раздела USING (эти столбцы должны быть одноименными в заголовках обеих таблиц). В обоих случаях типы данных каждой пары соответствующих столбцов должны быть совместимыми;
- будем называть списком выборки соответствующих столбцов соединения (select_list of corresponding join columns – SLCC) список элементов вида COALESCE (table1.c, table2.c) AS c*, где **c** является именем соответствующего столбца соединения. Элементы располагаются в том порядке, в котором они появляются в заголовке таблицы table1. Обозначим через SLT1 (SLT2) список имен столбцов таблицы table1 (table2), которые не являются соответствующими столбцами соединения. Имена располагаются в том же порядке, в котором они появляются в заголовке соответствующей таблицы;

заголовок результата совпадает с заголовком результата запроса

```
SELECT SLCC, SLT1, SLT2
FROM table1, table2;
```

# СОЕДИНЕННЫЕ ТАБЛИЦЫ: НАБОР СТРОК РЕЗУЛЬТАТА (ОБОЗНАЧЕНИЯ)

**Набор строк результата** (множество или мультимножество) определяется по следующим правилам. Обозначим через **T** следующие наборы строк:

- если видом соединения является UNION JOIN, то T – пусто;
- если видом соединения является CROSS JOIN, то T включает все строки, входящие в CP;
- если в спецификацию вида соединения входит раздел ON, то T включает все строки CP, для которых результатом вычисления условного выражения является true;
- если в спецификацию вида соединения входят разделы NATURAL или USING, и список SLCC не является пустым, то T включает все строки CP, для которых значения соответствующих столбцов соединения совпадают;
- если в спецификацию вида соединения входят разделы NATURAL или USING, и список SLCC является пустым, то T включает все строки CP.

- Обозначим через P1 (P2) набор (множество или мультимножество) всех строк таблицы table1 (table2), каждая из которых участвует в образовании некой строки T.
- Обозначим через U1 (U2) набор (множество или мультимножество) всех строк таблицы table1 (table2), ни одна из которых не участвует в образовании какой-либо строки T.
- Обозначим через X1 набор (множество или мультимножество) всех строк, образуемых из строк набора U1 путем добавления справа подстроки из неопределенных значений, содержащей столько неопределенных значений, сколько столбцов содержит таблица table2. Обозначим через X2 набор (множество или мультимножество) всех строк, образуемых из строк набора U2 путем добавления слева подстроки из неопределенных значений, содержащей столько неопределенных значений, сколько столбцов содержит таблица table1.

# СОЕДИНЕННЫЕ ТАБЛИЦЫ: НАБОР СТРОК РЕЗУЛЬТАТА

Для соединений вида CROSS JOIN и INNER JOIN пусть **S** обозначает тот же набор строк, что и T.

Для соединений вида LEFT OUTER JOIN пусть **S** обозначает набор строк, являющийся результатом выражения запросов

```
SELECT * FROM T
UNION ALL
SELECT * FROM X1;
```

Для соединений вида RIGHT OUTER JOIN пусть **S** обозначает набор строк, являющийся результатом выражения запросов

```
SELECT * FROM T
UNION ALL
SELECT * FROM X2;
```

Для соединений вида FULL OUTER JOIN пусть **S** обозначает набор строк, являющийся результатом выражения запросов

```
SELECT * FROM T
UNION ALL
SELECT * FROM X1
UNION ALL
SELECT * FROM X2;
```

Для соединений вида UNION JOIN пусть **S** обозначает набор строк, являющийся результатом выражения запросов

```
SELECT * FROM X1
UNION ALL
SELECT * FROM X2;
```

Если в спецификации вида соединения присутствуют ключевые слова NATURAL или USING, то результат операции совпадает с результатом выражения запросов

```
SELECT SLCC, SLT1, SLT2
FROM S;
```

Во всех остальных случаях результат операции совпадает с **S**.

# INNER JOIN ПО ОДНОМУ СТОЛБЦУ

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

Через **JR** будем обозначать результат соединения таблиц.

**JR**: table1 INNER JOIN table2 ON a1=b1 AND a2<b2 (внутреннее соединение по условию)

**JR**

| a1 | a2 | table1.c1 | table1.c2 | b1 | b2 | table2.c1 | table2.c2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |

**JR:** table1 INNER JOIN table2 USING (c2) (внутреннее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |

Строки-дубликаты появились в JR, поскольку в первом операнде присутствовали строки-дубликаты, удовлетворяющие условию соединения.

# INNER JOIN ПО НЕСКОЛЬКИМ СТОЛБЦАМ

**JR:** table1 INNER JOIN table2 USING (c1,c2)

**JR**

| a1 | a2 | c1 | c2 | b1 | b2 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |

Такой же результат будет получен при выполнении операции table1 NATURAL INNER JOIN table2 (естественное внутреннее соединение). Более того, для произвольных таблиц table1 и table2 результаты операций table1 INNER JOIN table2 USING (c1, c2, ...cn) и table1 INNER NATURAL JOIN table2 совпадают в том и только в том случае, когда список имен столбцов c1, c2, ...cn включает все имена столбцов, общие для таблиц table1 и table2.

**JR:** table1 LEFT OUTER JOIN table2 ON a1=b1 AND a2<b2 (левое внешнее соединение по условию>)

**JR**

| a1 | a2 | table1.c1 | table1.c2 | b1 | b2 | table2.c1 | table2.c2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL | NULL |
| 3 | NULL | NULL | 5 | NULL | NULL | NULL | NULL |

table 1

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

table 2

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

# RIGHT OUTER JOIN ПО УСЛОВИЮ

**JR:** table1 RIGHT OUTER JOIN table2 ON a1=b1 AND a2<b2 (правое внешнее соединение по условию)

**JR**

table 2

| a1 | a2 | table1.c1 | table1.c2 | b1 | b2 | table2.c1 | table2.c2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| NULL | NULL | NULL | NULL | 1 | 1 | 1 | 1 |
| NULL | NULL | NULL | NULL | 3 | 3 | 2 | 3 |
| NULL | NULL | NULL | NULL | 4 | 4 | 4 | 4 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

# FULL OUTER JOIN ПО УСЛОВИЮ

**JR:** table1 FULL OUTER JOIN table2 ON a1=b1 AND a2<b2 (полное внешнее соединение по условию)

table 1

**JR**

| a1 | a2 | table1.c1 | table1.c2 | b1 | b2 | table2.c1 | table2.c2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL | NULL |
| 3 | NULL | NULL | 5 | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | 1 | 1 | 1 | 1 |
| NULL | NULL | NULL | NULL | 3 | 3 | 2 | 3 |
| NULL | NULL | NULL | NULL | 4 | 4 | 4 | 4 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

Остальные строки table 2, для которых условие не выполнено

Как видно, в результате полного внешнего соединения сохраняются данные обоих операндов. Кстати, полное внешнее соединение иногда называют еще **симметричным внешним соединением**. Все операции внутреннего соединения и операция полного внешнего соединения коммутативны, а операции левого и правого соединения коммутативными не являются.

# LEFT OUTER JOIN

**JR:** table1 LEFT OUTER JOIN table2 USING (c2) (левое внешнее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

# RIGHT OUTER JOIN

**JR:** table1 RIGHT OUTER JOIN table2 USING (c2) (правое внешнее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| NULL | NULL | NULL | 4 | 4 | 4 | 4 |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

# RIGHT И LEFT OUTER JOIN

**JR:** table1 **LEFT** OUTER JOIN table2 USING (c2) (левое внешнее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL |

**JR:** table1 **RIGHT** OUTER JOIN table2 USING (c2) (правое внешнее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| NULL | NULL | NULL | 4 | 4 | 4 | 4 |

Отличаются значениями, для которых не нашлось соответствия

# FULL OUTER JOIN

**JR:** table1 **FULL** OUTER JOIN table2 USING (c2) (полное внешнее соединение по совпадению значений указанных одноименных столбцов)

**JR**

| a1 | a2 | table1.c1 | c2 | b1 | b2 | table2.c1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 | 2 |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 3 | NULL | NULL | 5 | 3 | NULL | NULL |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | 4 | 4 | 4 | 4 |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

В конец добавляем строки обеих таблиц, для которых не нашлось соответствия

# NATURAL LEFT OUTER JOIN

**JR:** table1 LEFT OUTER JOIN table2 USING (c2, c1) (и операции table1 NATURAL LEFT OUTER JOIN table2 – естественное левое внешнее соединение)

**JR**

| a1 | a2 | c1 | c2 | b1 | b2 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 2 | 3 | 4 | NULL | NULL | NULL |
| 3 | NULL | NULL | 5 | NULL | NULL |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

Перебираем пары (c1,c2) из table 1. NULL ни с чем не сравниваем.

# NATURAL RIGHT OUTER JOIN

**JR:** table1 RIGHT OUTER JOIN table2 USING (c2, c1) (и операции table1 NATURAL RIGHT OUTER JOIN table2 – естественное правое внешнее соединение)

**JR**

| a1 | a2 | c1 | c2 | b1 | b2 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| NULL | NULL | 4 | 4 | 4 | 4 |
| NULL | NULL | NULL | 5 | 3 | NULL |
| NULL | NULL | NULL | 5 | 3 | NULL |

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

Перебираем пары (c1,c2) из table 2.

# NATURAL FULL OUTER JOIN

**JR:** table1 FULL OUTER JOIN table2 USING (c2, c1) (и операции table1 NATURAL FULL OUTER JOIN table2 – естественное полное внешнее соединение)

**JR**

| a1 | a2 | c1 | c2 | b1 | b2 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 1 | 1 | 2 | 3 | 1 | 2 |
| 1 | 1 | 2 | 3 | 3 | 3 |
| 2 | 3 | 4 | NULL | NULL | NULL |
| 3 | NULL | NULL | 5 | NULL | NULL |
| NULL | NULL | 4 | 4 | 4 | 4 |
| NULL | NULL | NULL | 5 | 3 | NULL |
| NULL | NULL | NULL | 5 | 3 | NULL |

table 1

table 2

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

Перебираем пары (c1,c2) из обеих таблиц.

# UNION JOIN

**JR:** table1 UNION JOIN table2 (соединение объединением)

**JR**

table 1

| a1 | a2 | table1.c1 | table1.c2 | b1 | b2 | table2.c1 | table2.c2 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | NULL | NULL | NULL | NULL |
| 1 | 1 | 2 | 3 | NULL | NULL | NULL | NULL |
| 1 | 1 | 2 | 3 | NULL | NULL | NULL | NULL |
| 2 | 3 | 4 | NULL | NULL | NULL | NULL | NULL |
| 3 | NULL | NULL | 5 | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | 1 | 1 | 2 | 3 |
| NULL | NULL | NULL | NULL | 1 | 2 | 2 | 3 |
| NULL | NULL | NULL | NULL | 3 | 3 | 2 | 3 |
| NULL | NULL | NULL | NULL | 4 | 4 | 4 | 4 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |
| NULL | NULL | NULL | NULL | 3 | NULL | NULL | 5 |

table 2

**table 1**

| a1 | a2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | NULL |
| 3 | NULL | NULL | 5 |

**table 2**

| b1 | b2 | c1 | c2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 3 | 2 | 3 |
| 4 | 4 | 4 | 4 |
| 3 | NULL | NULL | 5 |
| 3 | NULL | NULL | 5 |

# ПРИМЕРЫ НА ОСНОВЕ БАЗЫ ДАННЫХ

EMP:

| EMP_NO : EM_NO |
|---|
| EMP_NAME : VARCHAR |
| EMP_BDATE : DATE |
| EMP_SAL : SALARY |
| DEPT_NO : DEPT_NO |
| PRO_NO : PRO_NO |

DEPT:

| DEPT_NO : DEPT_NO |
|---|
| DEPT_NAME : VARCHAR |
| DEPT_EMP_NO : INTEGER |
| DEPT_TOTAL_SAL : SALARY |
| DEPT_MNG : EMP_NO |

PRO:

| PRO_NO : PRO_NO |
|---|
| PRO_TITLE : VARCHAR |
| PRO_SDATE : DATEP |
| PRO_DURAT : INTERVAL |
| PRO_MNG : EMP_NO |
| PRO_DESC : CLOB |

Столбцы EMP_NO, DEPT_NO и PRO_NO являются первичными ключами таблиц EMP, DEPT и PRO соответственно. Столбцы DEPT_NO и PRO_NO таблицы EMP являются внешними ключами, ссылающимися на таблицы DEPT и PRO соответственно (DEPT_NO указывает на отделы, в которых работают служащие, а PRO_NO – на проекты, в которых они участвуют; оба столбца могут принимать неопределенные значения). Столбец DEPT_MNG является внешним ключом таблицы DEPT (DEPT_MNG указывает на служащих, которые исполняют обязанности руководителей отделов; у отдела может не быть руководителя, и один служащий не может быть руководителем двух или более отделов). Столбец PRO_MNG является внешним ключом таблицы PRO (PRO_MNG указывает на служащих, которые являются менеджерами проектов, у проекта всегда есть менеджер, и один служащий не может быть менеджером двух или более проектов).

# ПРИМЕРЫ

**ПРИМЕР 1** Для каждого отдела найти его номер, имя руководителя, число служащих, минимальный, максимальный и средний размеры зарплаты служащих

```
SELECT DEPT.DEPT_NO, EMP1.EMP_NAME, COUNT(*), MIN(EMP2.EMP_SAL),
   MAX(EMP2.EMP_SAL), AVG(EMP2.EMP_SAL)
FROM (DEPT NATURAL INNER JOIN EMP AS EMP2)
   INNER JOIN EMP AS EMP1 ON DEPT.DEPT_MNG = EMP1.EMP_NO
GROUP BY DEPT.DEPT_NO, EMP1.EMP_NAME;
```

**ПРИМЕР 2** Найти номера служащих и имена их начальников отделов для служащих, размер зарплаты которых больше 30000 руб.

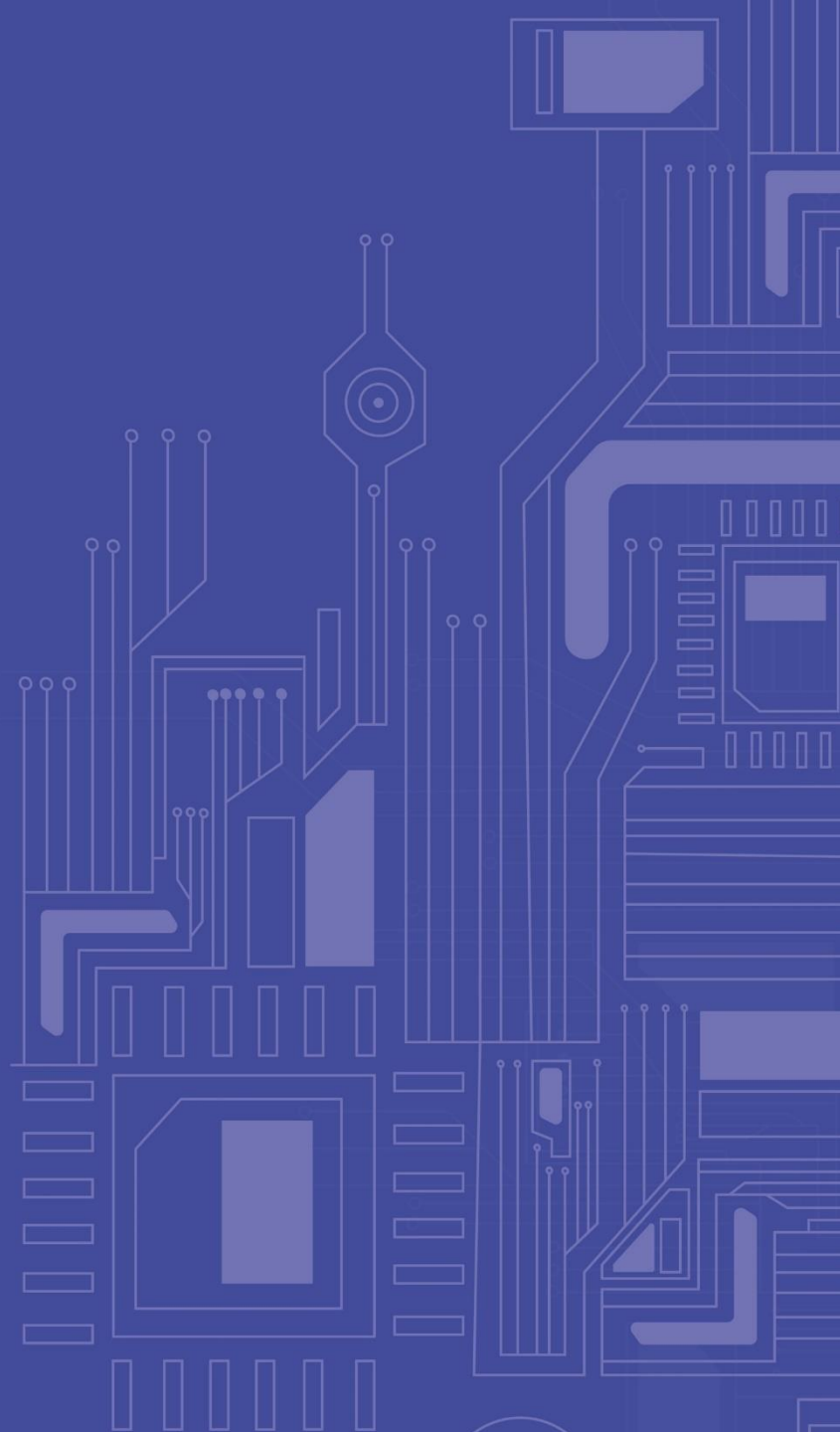
```
SELECT EMP1.EMP_NO, EMP2.EMP_NAME
FROM (EMP AS EMP1 NATURAL INNER JOIN DEPT)
   INNER JOIN EMP AS EMP2 ON DEPT.DEPT_MNG = EMP2.EMP_NO
WHERE EMP1.EMP_SAL > 30000.00;
```

<=>

```
SELECT EMP1.EMP_NO, EMP2.EMP_NAME
FROM (EMP AS EMP1 INNER JOIN DEPT
   ON EMP1.DEPT_NO = DEPT.DEPT_NO AND
     EMP1.EMP_SAL > 30000.00)
   INNER JOIN EMP AS EMP2 ON DEPT.MNG = EMP2.EMP_NO;
```

Можно обойтись вообще без раздела WHERE, если пожертвовать «естественностью» первого соединения

# АНАЛИТИЧЕСКИЕ ЗАПРОСЫ



# АНАЛИТИЧЕСКИЕ ЗАПРОСЫ

*Аналитическими запросами к базе данных* принято называть запросы, сводные (агрегатные) результаты которых вычисляются над детальными данными, хранящимися в таблицах базы данных. В этом смысле любой запрос на языке SQL, результат которого основан на вычислении агрегатных функций, можно назвать аналитическим. Характерная особенность аналитических запросов состоит в том, что, как правило, они применяются к большим по объему базам данных, и выполнение таких запросов вызывает существенные накладные расходы СУБД.

## ПРИМЕР

| EMP_NO | DEPT_NO | EMP_BDATE | EMP_SAL |
|---|---|---|---|
| 2440 | 1 | 1950 | 15000.00 |
| 2441 | 1 | 1950 | 16000.00 |
| 2442 | 1 | 1960 | 14000.00 |
| 2443 | 1 | 1960 | 19000.00 |
| 2444 | 2 | 1950 | 17000.00 |
| 2445 | 2 | 1950 | 16000.00 |
| 2446 | 2 | 1960 | 14000.00 |
| 2447 | 2 | 1960 | 20000.00 |
| 2448 | 3 | 1950 | 18000.00 |
| 2449 | 3 | 1950 | 13000.00 |
| 2450 | 3 | 1960 | 21000.00 |
| 2451 | 3 | 1960 | 22000.00 |

Требуется узнать максимальный размер зарплаты на всем предприятии, максимальный размер зарплаты в каждом отделе и максимальный размер зарплаты служащих каждой возрастной категории каждого отдела.

```
SELECT MAX (EMP_SAL) AS MAX_ENT_SAL
    FROM EMP;

SELECT DEPT_NO, MAX (EMP_SAL) AS MAX_DEP_SAL
    FROM EMP
    GROUP BY DEPT_NO;

SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL)
    AS MAX_DEP_BDATE_SAL
    FROM EMP
    GROUP BY DEPT_NO, EMP_BDATE;
```

| MAX_ENT_SAL |
|---|
| 22000.00 |

| DEPT_NO | MAX_DEP_SAL |
|---|---|
| 1 | 19000.00 |
| 2 | 20000.00 |
| 3 | 22000.00 |

| DEPT_NO | EMP_BDATE | MAX_DEP_BDATE_SAL |
|---|---|---|
| 1 | 1950 | 16000.00 |
| 1 | 1960 | 19000.00 |
| 2 | 1950 | 17000.00 |
| 2 | 1960 | 20000.00 |
| 3 | 1950 | 18000.00 |
| 3 | 1960 | 22000.00 |

# GROUP BY ROLLUP

Эти же результаты можно получить при выполнении единственного запроса, если в его формулировке использовать специальный вид группировки ROLLUP:

```
SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL) AS MAX_SAL
    FROM EMP
    GROUP BY ROLLUP (DEPT_NO, EMP_BDATE);
```

| DEPT_NO | EMP_BDATE | MAX_SAL |
|---|---|---|
| NULL | NULL | 22000.00 |
| 1 | NULL | 19000.00 |
| 2 | NULL | 20000.00 |
| 3 | NULL | 22000.00 |
| 1 | 1950 | 16000.00 |
| 1 | 1960 | 19000.00 |
| 2 | 1950 | 17000.00 |
| 2 | 1960 | 20000.00 |
| 3 | 1950 | 18000.00 |
| 3 | 1960 | 22000.00 |

В столбце MAX_SAL первой строки результирующей таблицы находится максимальное значение зарплаты служащих на всем предприятии. Столбцы DEPT_NO и EMP_BDATE в этой строке содержат неопределенное значение, поскольку значение MAX_SAL не привязано к каким-либо отделу и возрастной категории. В столбце MAX_SAL следующих трех строк находятся максимальные значения зарплаты служащих отделов с номерами 1, 2 и 3 соответственно, что показывают значения столбца DEPT_NO. Столбец EMP_BDATE в этих строках содержит неопределенное значение, поскольку значение MAX_SAL не привязано к какой-либо возрастной категории. Наконец, в столбце MAX_SAL в последних шести строках содержатся максимальные значения зарплаты служащих каждой возрастной категории каждого отдела, что показывают значения столбцов DEPT_NO и EMP_BDATE, которые теперь содержат соответствующий номер отдела и год рождения служащих.

# ТЕОРИЯ GROUP BY ROLLUP

В общем случае пусть раздел группир$n$), где $cname_i$ ($i$ = 1, 2, ... , $n$) – имя столбца таблицы-результата раздела FROM запроса. Пусть в списке выборки используются вызовы агрегатных функций $AGG_1$, $AGG_2$, ... , $AGG_m$ над значениями столбцов, не входящих в список группировки, а также имена столбцов $cname_1$, $cname_2$, ... , $cname_n$. Тогда запрос выполняется следующим образом. Первая строка результата (первый набор строк результирующей таблицы) производится таким образом, как если бы в запросе вообще отсутствовал раздел GROUP BY, т.е. агрегатные функции $AGG_1$, $AGG_2$, ... , $AGG_m$ вычисляются над значениями всех строк таблицы. Значением столбцов $cname_1$, $cname_2$, ... , $cname_n$ в этой строке является NULL. ($i$+1)-й набор строк результата формируется так, как если бы раздел группировки запроса имел вид GROUP BY ($cname_1$, $cname_2$, ... , $cname_i$) ($1 \le i < n$). Во всех этих строках значением столбцов $cname_{(i+1)}$, ... , $cname_n$ является NULL. Наконец, ($n$+1)-й набор строк результата формируется так, как если бы раздел группировки запроса имел вид GROUP BY ($cname_1$, $cname_2$, ... , $cname_n$).

Может показаться, что запросы, содержащие раздел GROUP BY ROLLUP, настолько сложны, что их выполнение будет занимать чрезмерно большое время. Это ощущение является ложным. В действительности, при выполнении запросов с обычной группировкой вида GROUP BY $cname_1$, $cname_2$, ... , $cname_n$, как правило, последовательно выполняется сортировка строк таблицы-результата раздела FROM в соответствии со значениями столбца $cname_1$, затем – в соответствии со значениями столбца $cname_2$ и т. д., и в заключение – сортировка в соответствии со значениями столбца $cname_n$. Во время выполнения каждой сортировки можно заодно вычислять значения агрегатных функций. Так что стоимость выполнения запроса, содержащего раздел GROUP BY ROLLUP, лишь незначительно отличается от стоимости выполнения запроса с обычной группировкой.

# ПРИМЕР ROLLUP БЕЗ GROUPING

ПРИМЕР

```
SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL) AS MAX_SAL
    FROM EMP
    GROUP BY ROLLUP (DEPT_NO, EMP_BDATE);
```

EMP

| EMP_NO | DEPT_NO | EMP_BDATE | EMP_SAL |
|---|---|---|---|
| 2440 | 1 | 1950 | 15000.00 |
| 2441 | 1 | 1950 | 16000.00 |
| 2442 | 1 | 1960 | 14000.00 |
| 2443 | 1 | 1960 | 19000.00 |
| 2452 | 1 | NULL | 15000.00 |
| 2453 | 1 | NULL | 17000.00 |
| 2444 | 2 | 1950 | 17000.00 |
| 2445 | 2 | 1950 | 16000.00 |
| 2446 | 2 | 1960 | 14000.00 |
| 2447 | 2 | 1960 | 20000.00 |
| 2448 | 3 | 1950 | 18000.00 |
| 2449 | 3 | 1950 | 13000.00 |
| 2450 | 3 | 1960 | 21000.00 |
| 2451 | 3 | 1960 | 22000.00 |
| 2454 | NULL | 1950 | 13000.00 |
| 2455 | NULL | 1950 | 14000.00 |
| 2456 | NULL | NULL | 19000.00 |

| DEPT_NO | EMP_BDATE | MAX_SAL |
|---|---|---|
| NULL | NULL | 22000.00 |
| NULL | NULL | 19000.00 |
| NULL | NULL | 14000.00 |
| 1 | NULL | 19000.00 |
| 2 | NULL | 20000.00 |
| 3 | NULL | 22000.00 |
| 1 | NULL | 17000.00 |
| 1 | 1950 | 16000.00 |
| 1 | 1960 | 19000.00 |
| 2 | 1950 | 17000.00 |
| 2 | 1960 | 20000.00 |
| 3 | 1950 | 18000.00 |
| 3 | 1960 | 22000.00 |
| NULL | 1950 | 14000.00 |

Невозможно установить, в какой из первых трех строк неопределенное значение столбцов DEPT_NO и EMP_BDATE означает то, что эта строка является сводной для всего предприятия, а не то, что она является сводной для всех служащих с неизвестными номером отдела и годом рождения или просто для всех служащих с неизвестным номером отдела. Аналогичным образом невозможно понять, какая строка в следующей далее паре строк является сводной для всех служащих отдела номер 1, а не для всех служащих отдела номер 1 с неизвестным годом рождения.

# АГРЕГАТНАЯ ФУНКЦИЯ GROUPING

Для того чтобы всегда можно было разобраться в результатах запросов, включающих раздел GROUP BY ROLLUP, в язык SQL была введена специальная *агрегатная функция GROUPING*. Эта функция применяется к столбцу, входящему в список столбцов раздела GROUP BY ROLLUP, и принимает целое значение 1 в тех строках результирующей таблицы, в которых соответствующий столбец имеет значение NULL по той причине, что строка является сводной для более обобщенной группы. В противном случае функция GROUPING принимает значение 0.

```
SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL) AS MAX_SAL,
    GROUPING (DEPT_NO) AS GDN, GROUPING (EMP_BDATE)
    AS GEB
    FROM EMP
    GROUP BY ROLLUP (DEPT_NO, EMP_BDATE);
```

| DEPT_NO | EMP_BDATE | MAX_SAL | GDN | GEB |
|---|---|---|---|---|
| NULL | NULL | 22000.00 | 1 | 1 |
| NULL | NULL | 19000.00 | 0 | 0 |
| NULL | NULL | 14000.00 | 1 | 0 |
| 1 | NULL | 19000.00 | 0 | 1 |
| 2 | NULL | 20000.00 | 0 | 1 |
| 3 | NULL | 22000.00 | 0 | 1 |
| 1 | NULL | 17000.00 | 0 | 0 |
| 1 | 1950 | 16000.00 | 0 | 0 |
| 1 | 1960 | 19000.00 | 0 | 0 |
| 2 | 1950 | 17000.00 | 0 | 0 |
| 2 | 1960 | 20000.00 | 0 | 0 |
| 3 | 1950 | 18000.00 | 0 | 0 |
| 3 | 1960 | 22000.00 | 0 | 0 |
| NULL | 1950 | 14000.00 | 0 | 0 |

То, что было, для сравнения:

```
SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL) AS MAX_SAL
    FROM EMP
    GROUP BY ROLLUP (DEPT_NO, EMP_BDATE);
```

# GROUP BY CUBE

В отличие от запросов с традиционной группировкой, результат запроса, содержащего раздел GROUP BY ROLLUP, зависит от порядка столбцов в списке группировки. При выполнении запроса происходит движение по этому списку слева направо с повышением уровня детальности результирующих данных. Существует еще одна разновидность запроса с группировкой, основанная на использовании раздела **GROUP BY CUBE**.

Пусть раздел группировки запроса имеет вид GROUP BY CUBE (cname1, cname2, ... , cname$n$), где cname$_i$ ($i$ = 1, 2, ... , $n$) – имя столбца таблицы-результата раздела FROM запроса. Обозначим через SGBC множество {cname$_1$, cname$_2$, ... , cname$_n$}. Пусть $S_i$ является произвольным подмножеством SGBC, т.е. $S_i$ представляет собой пустое множество или имеет вид {cname$_{i1}$, cname$_{i2}$, ... , cname$_{im}$}, где $m \leq n$, и каждое имя столбца cname$_{ij}$ совпадает с одним и только одним именем столбца из списка столбцов раздела GROUP BY CUBE. Очевидно, что у множества SGBC существует $2^n$ подмножеств различных вида $S_i$. Тогда по определению результат этого запроса совпадает с объединением результатов $2^n$ запросов с теми же разделами SELECT, FROM и WHERE, что и у запроса с GROUP BY CUBE, и с разделом группировки вида GROUP BY $S_i$, причем во всех строках результата частичного запроса значением любого столбца cname$_j$ такого, что cname$_j$ $\in$ SGBC и cname$_j$ $\notin$ $S_i$, является NULL. Запрос с разделом группировки вида GROUP BY S, где S – пустое множество, трактуется как запрос без раздела GROUP BY.

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |

# ПРИМЕР GROUP BY CUBE

**ПРИМЕР** Найти максимальный размер зарплаты во всем предприятии, максимальный размер зарплаты в каждом отделе, максимальный размер зарплаты служащих в каждой возрастной категории и максимальный размер зарплаты служащих каждой возрастной категории каждого отдела.

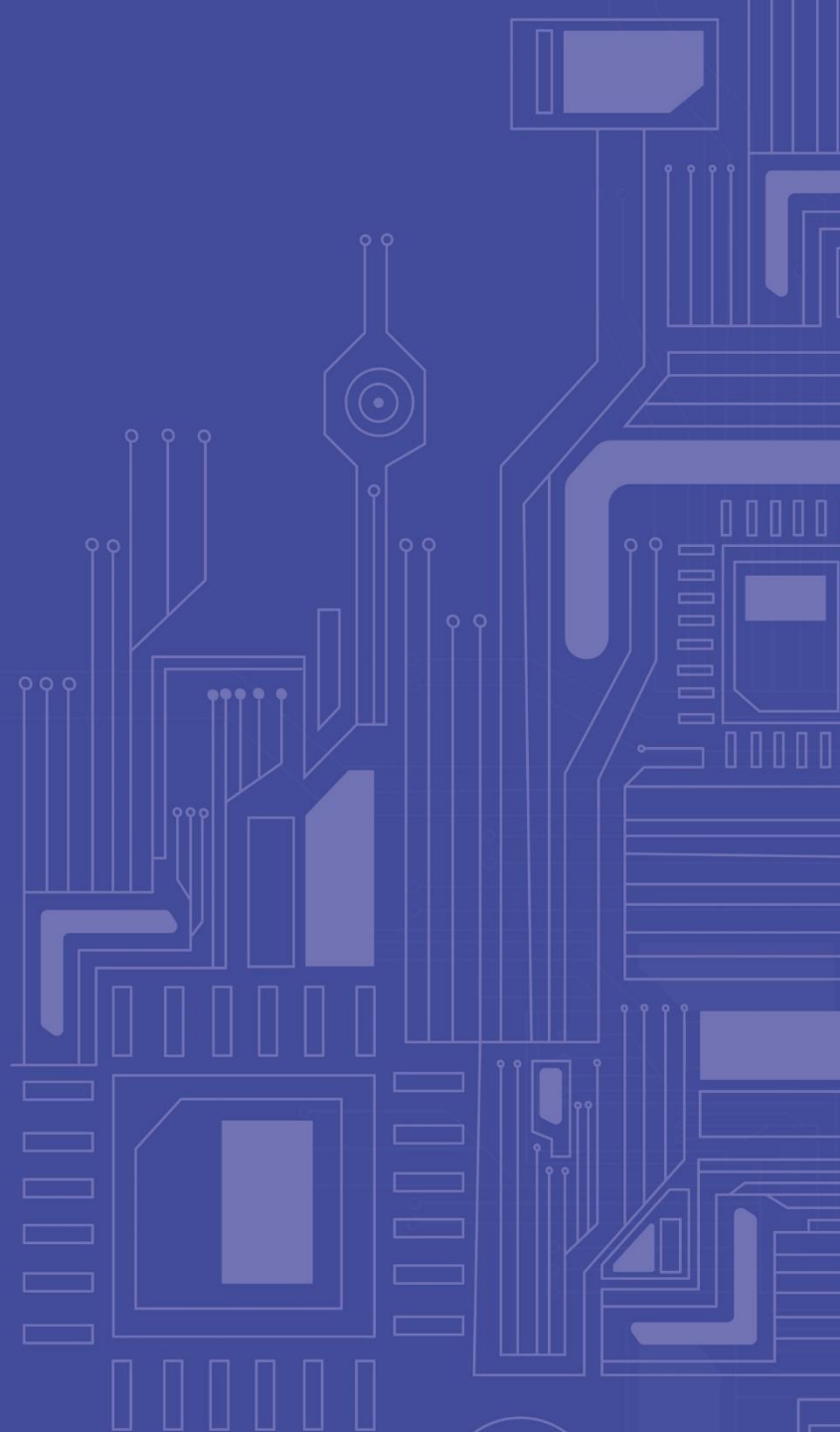
```
SELECT DEPT_NO, EMP_BDATE, MAX (EMP_SAL)AS MAX_SAL,
GROUPING (DEPT_NO) AS GDN, GROUPING (EMP_BDATE) AS GEB
FROM EMP
GROUP BY CUBE (DEPT_NO, EMP_BDATE);
```

| DEPT_NO | EMP_BDATE | MAX_SAL | GDN | GEB |
|---|---|---|---|---|
| NULL | NULL | 22000.00 | 1 | 1 |
| NULL | NULL | 19000.00 | 0 | 0 |
| NULL | NULL | 14000.00 | 1 | 0 |
| 1 | NULL | 19000.00 | 0 | 1 |
| 2 | NULL | 20000.00 | 0 | 1 |
| 3 | NULL | 22000.00 | 0 | 1 |
| 1 | NULL | 17000.00 | 0 | 0 |
| 1 | 1950 | 16000.00 | 0 | 0 |
| 1 | 1960 | 19000.00 | 0 | 0 |
| 2 | 1950 | 17000.00 | 0 | 0 |
| 2 | 1960 | 20000.00 | 0 | 0 |
| 3 | 1950 | 18000.00 | 0 | 0 |
| 3 | 1960 | 22000.00 | 0 | 0 |
| NULL | 1950 | 14000.00 | 0 | 0 |
| NULL | 1950 | 18000.00 | 1 | 0 |
| NULL | 1960 | 22000.00 | 1 | 0 |

Добавились две последние строки, показывающие максимальные значения зарплаты всех служащих предприятия, родившихся в 1950-м и 1960-м гг. соответственно.

Наш пример может навести на мысль, что и в общем случае запросы, содержащие раздел GROUP BY CUBE, не слишком отличаются от запросов с GROUP BY ROLLUP, и выполнение этих запросов тоже не слишком различается. Однако это совсем не так. Запрос, содержащий раздел GROUP BY CUBE, действительно вырождается в объединение результатов $2^n$ запросов с обычным разделом GROUP BY. Соответственно, сложность выполнения такого запроса несравненно выше сложности выполнения похожего запроса с GROUP BY ROLLUP. В нашем примере все получилось так просто только по той причине, что в запросе имеются всего два столбца группировки.

# БАЗОВЫЕ СРЕДСТВА МАНИПУЛИРОВАНИЯ ДАННЫМИ



# БАЗОВЫЕ СРЕДСТВА DML

К базовым средствам манипулирования данными языка SQL относятся «поисковые» варианты операторов **UPDATE** и **DELETE**. Эти варианты называются поисковыми, потому что при задании соответствующей операции задается логическое условие, налагаемое на строки адресуемой оператором таблицы, которые должны быть подвергнуты модификации или удалению. Кроме того, в такую категорию языковых средств входит оператор **INSERT**, позволяющий добавлять строки в существующие таблицы.

## Оператор INSERT для вставки строк в существующие таблицы

Общий синтаксис оператора INSERT выглядит следующим образом:

```
INSERT INTO table_name
     { [ (column_commalist) ] query_expression
     | DEFAULT VALUES
```

Даже если ограничиться простейшей составляющей этой конструкции (simple_table), то мы имеем следующие возможности:

```
simple_table ::= query_specification
     | table_value_constructor
     | TABLE table_name
```

```
query_expression ::= [ with_clause ] query_expression_body
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
query_term ::= non_join_query_term | joined_table
non_join_query_term ::= non_join_query_primary
   | query_term INTERSECT [ ALL | DISTINCT ]
     [ corresponding_spec ] query_primary
query_primary ::= non_join_query_primary | joined_table
non_join_query_primary ::= simple_table
   | (non_join_query_expression)
simple_table ::= query_specification
   | table_value_constructor
   | TABLE table_name
corresponding_spec ::= CORRESPONDING
   [ BY column_name_comma_list ]
```

# ВСТАВКА ВСЕХ СТРОК УКАЗАННОЙ ТАБЛИЦЫ

Стандарт допускает вставку в указанную таблицу всех строк некоторой другой таблицы (вариант table_name). Эта другая таблица может быть как базовой, так и представляемой.

- В определении представления не должны присутствовать ссылки на таблицу, в которую производится вставка.
- При использовании данного варианта оператора вставки число столбцов вставляемой таблицы должно совпадать с числом столбцов таблицы, в которую производится вставка, или с числом столбцов, указанных в списке column_commalist, если этот список задан.
- Типы данных соответствующих столбцов вставляемой таблицы и таблицы, в которую производится вставка, должны быть совместимыми.
- Если в операции задан список column_commalist и в нем содержатся не все имена столбцов таблицы, в которую производится вставка, то в оставшиеся столбцы во всех строках заносятся значения столбцов по умолчанию.
- Если для какого-либо из оставшихся столбцов значение по умолчанию не определено, при выполнении операции вставки фиксируется ошибка.

**ПРИМЕР** Предположим, что в базе данных EMP-DEPT-PRO имеется еще одна промежуточная таблица EMP_TEMP, в которой временно хранятся данные о служащих, проходящих испытательный срок. Пусть эта таблица имеет следующий заголовок:

EMP_TEMP:

| EMP_NO : EMP_NO |
|---|
| EMP_NAME : VARCHAR |
| EMP_BDATE : DATE |

```
INSERT INTO EMP (EMP_NO, EMP_NAME, EMP_BDATE) TABLE EMP_TEMP;
```

В столбцах EMP_NO, EMP_NAME, EMP_BDATE будут данные, взятые из таблицы EMP_TEMP, а в столбцах EMP_SAL, DEPT_NO, PRO_NO будут значения по умолчанию.

EMP_NO не должен нарушать ограничения целостности первичного ключа.

# ВСТАВКА ЯВНО ЗАДАННОГО ЧИСЛА СТРОК

Набор вставляемых строк задается явно с использованием синтаксической конструкции table_value_constructor.

```
table_value_constructor ::=
    VALUES row_value_constructor_comma_list
row_value_constructor ::= row_value_constructor_element
    | [ ROW ] (row_value_constructor_element_comma_list)
    | row_subquery
row_value_constructor_element ::= value_expression
    | NULL | DEFAULT
```

**Варианты вставки:**

```
INSERT INTO EMP
   ROW (2445, 'Brown', '1985-04-08', 16500.00, 630, 772);
```

```
INSERT INTO EMP
   ROW ( 2445, DEFAULT, NULL, DEFAULT, NULL, NULL);
```

<=>

```
INSERT INTO EMP (EMP_NO) 2445;
```

Одной из разновидностей value_expression_primary является **scalar_subquery**

```
INSERT INTO EMP VALUES
           ROW (2445, (SELECT EMP_NAME
                       FROM EMP
                       WHERE EMP_NO = 2555),
                    '1985-04-08',
                    SELECT EMP_SAL
                       FROM EMP
                       WHERE EMP_NO = 2555),
                    NULL, NULL ),
           ROW (2446, (SELECT EMP_NAME
                       FROM EMP
                       WHERE EMP_NO = 2556),
                    '1978-05-09',
                    (SELECT EMP_SAL
                       FROM EMP
                       WHERE EMP_NO = 2556),
                    NULL, NULL );
```

После выполнения этой операции в таблице EMP появятся две новые строки для служащих с уникальными идентификаторами 2445 и 2446, причем первому из них будет присвоено имя и размер заработной платы служащего с уникальным идентификатором 2555, а второму – аналогичные данные о служащем с уникальным идентификатором 2556.

# ВСТАВКА СТРОК РЕЗУЛЬТАТА ЗАПРОСА

Это вариант оператора вставки, когда набор вставляемых строк определяется через спецификацию запроса.

**ПРИМЕР** Требуется сохранить в отдельной таблице DEPT_SUMMARY сведения о числе служащих каждого отдела, их максимальной, минимальной и суммарной заработной плате.

DEPT_SUMMARY:

| |
|---|
| DEPT_NO : DEPT_NO |
| DEPT_EMP_NO : INTEGER |
| DEPT_MAX_SAL : SALARY |
| DEPT_MIN_SAL : SALARY |
| DEPT_TOTAL_SAL : SALARY |

```
INSERT INTO DEPT_SUMMARY
   (SELECT DEPT_NO, COUNT(*), MAX (EMP_SAL),
        MIN (EMP_SAL), SUM (EMP_SAL)
     FROM EMP
     GROUP BY DEPT_NO);
```

# ОПЕРАТОР UPDATE

Общий синтаксис оператора UPDATE выглядит следующим образом:

```
UPDATE table_name SET update_assignment_commalist
    WHERE conditional_expression
update_assignment ::= column_name =
    { value_expression | DEFAULT | NULL }
```

Как это работает:

1. Для всех строк таблицы с именем table_name вычисляется булевское выражение conditional_expression. Строки, для которых значением этого булевского выражения является true, считаются подлежащими модификации (обозначим множество таких строк через $T_m$);
2. Каждая строка s (из $T_m$) подвергается модификации таким образом, что значение каждого столбца этой строки, указанного в списке *update_assignment_commalist*, заменяется значением, указанным в правой части соответствующего элемента списка модификации. Значения столбцов строки s, не указанные в списке модификации, остаются неизменными.

## ПРИМЕР

Для всех служащих, работающих в отделах, заработная плата менеджеров которых превышает 30000 руб., установить размер заработной платы, на 1000 руб. превышающий средний размер заработной платы соответствующего отдела, а номера проектов, в которых участвуют эти служащие, сделать неопределенными.

```
UPDATE EMP SET EMP_SAL = (SELECT AVG (EMP1_SAL)
     FROM EMP EMP1
     WHERE EMP.DEPT_NO = EMP1.DEPT_NO)
         + 1000.00, PRO_NO = NULL
     WHERE (SELECT EMP1.EMP_SAL
           FROM EMP EMP1, DEPT
           WHERE EMP.DEPT_NO = DEPT.DEPT_NO
              AND DEPT_MNG = EMP1.EMP_NO AND) > 30000.00;
```

# ОПЕРАТОР DELETE

Общий синтаксис оператора UPDATE выглядит следующим образом:

```
DELETE FROM table_name
     WHERE conditional_expression
```

В некотором смысле оператор DELETE является частным случаем оператора UPDATE (или, наоборот, действие оператора UPDATE представляет собой комбинацию действий операторов DELETE и INSERT).

Как это работает:

1. Для всех строк таблицы с именем table_name вычисляется булевское выражение conditional_expression. Строки, для которых значением этого булевского выражения является true, считаются подлежащими удалению (обозначим множество таких строк через $T_d$);
2. Каждая строка s (из $T_d$) удаляется из указанной таблицы.

**ПРИМЕР 1** Удалить из таблицы EMP все строки, относящиеся к служащим, которые участвуют в проекте с номером 772.

```
DELETE FROM EMP WHERE PRO_NO = 772;
```

**ПРИМЕР 2** Удалить из таблицы EMP все строки, относящиеся к служащим, размер заработной платы которых превышает размер заработной платы менеджеров их отделов.

```
DELETE FROM EMP WHERE EMP_SAL >
     (SELECT EMP1.EMP_SAL
          FROM EMP EMP1, DEPT
          WHERE EMP.DEPT_NO = DEPT.DEPT_NO
             AND DEPT.DEPT.MNG = EMP1.EMP_NO);
```


ВСЁ!

# ПРИМЕРЫ ЗАПРОСОВ

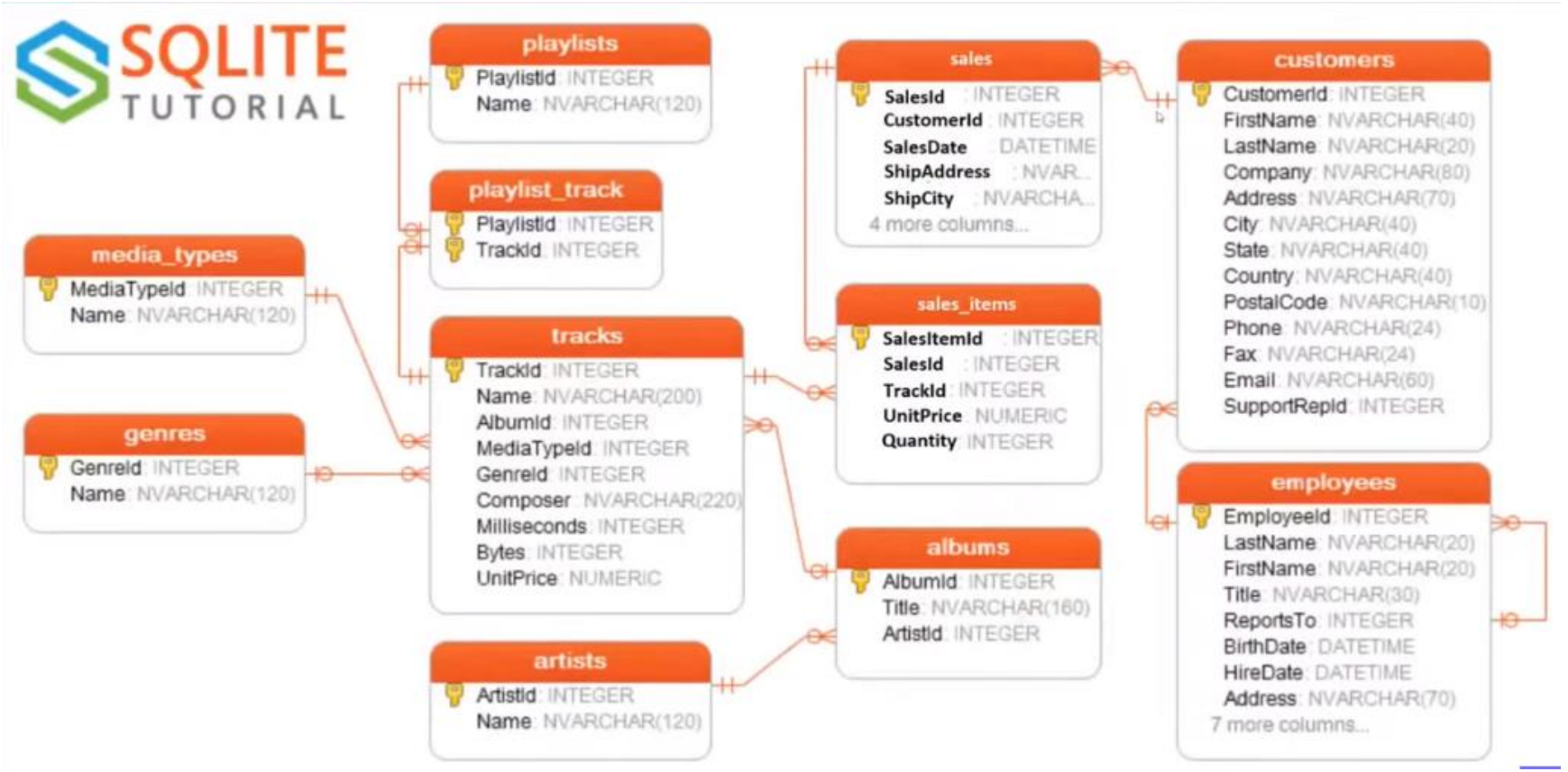
```sql
select ShipCity,count(*) as cnt
from sales
 where   ShipCountry='Germany'
     and SalesDate >= date('2009-01-01')
     and SalesDate < date('2010-01-01')
group by  ShipCity
order by ShipCity desc limit 2
```

```sql
UNION
    ALL --полный
    DISTINCT --исключающий

ПОДЗАПРОСЫ
    коррелированый
    без корреляции

JOIN
    CROSS
    INNER
    OUTER
        LEFT
        RIGHT
        FULL

EXCEPT -- логическое вычитание
INTERSECT -- логическое пересечение
```




В SQLite есть только LEFT OUTER JOIN, других нет, но он может все заменить.

# ЗАПРОСЫ С UNION, EXCEPT И INTERSECT

```
select FirstName,LastName, 'Клиент' as  client_type from customers
union all
select  FirstName,LastName, 'Сотрудник'  from  employees
```

Для UNION ALL две таблицы должны быть одинаковой структуры

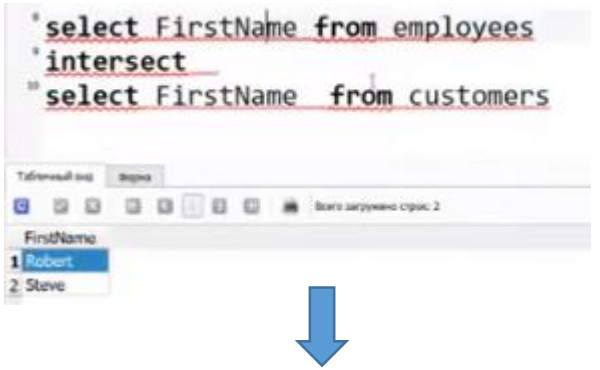
| | FirstName | LastName | client_type |
|---|---|---|---|
| 54 | Steve | Murray | Клиент |
| 55 | Mark | Taylor | Клиент |
| 56 | Diego | Gutiérrez | Клиент |
| 57 | Luis | Rojas | Клиент |
| 58 | Manoj | Pareek | Клиент |
| 59 | Puja | Srivastava | Клиент |
| 60 | Cage | Jonny | Клиент |
| 61 | Harvey | Queen | Клиент |
| 62 | Aaron | Lennon | Клиент |
| 63 | David | Beckham | Клиент |
| 64 | Andrew | Adams | Сотрудник |
| 65 | Nancy | Edwards | Сотрудник |
| 66 | Jane | Peacock | Сотрудник |
| 67 | Margaret | Park | Сотрудник |
| 68 | Steve | Johnson | Сотрудник |

```
select FirstName  from customers
union
select FirstName  from employees
```

| | FirstName |
|---|---|
| 1 | Aaron |
| 2 | Alexandre |
| 3 | Andrew |
| 4 | Astrid |
| 5 | Bjørn |
| 6 | Cage |
| 7 | Camille |
| 8 | Daan |

Отбрасывает дубликаты

```
select FirstName from employees
intersect
select FirstName  from customers
```



| | FirstName |
|---|---|
| 1 | Robert |
| 2 | Steve |

Показывает дубликаты

```
select FirstName  from customers
except
select FirstName  from employees
```

Имена только тех клиентов, которые не совпадают с именами сотрудников

Типы полей, порядок полей, количество полей должны быть одинаковыми в операндах EXCEPT, INTERSECT, UNION

# ЗАПРОСЫ С КОРРЕЛЯЦИЕЙ И БЕЗ

```
select  FirstName,LastName, (
        select count(*)
        from sales
    )
from customers
```

| | FirstName | LastName | (SELECT count(*) FROM sales) |
|---|---|---|---|
| 16 | Frank | Harris | 412 |
| 17 | Jack | Smith | 412 |
| 18 | Michelle | Brooks | 412 |
| 19 | Tim | Goyer | 412 |
| 20 | Dan | Miller | 412 |
| 21 | Kathy | Chase | 412 |

Запрос без корреляции, связи между таблицами не учитываются, каждому клиенту дописали общее число продаж.

```
select  FirstName,LastName, (
        select count(*)
        from sales as s
        where  s.CustomerId= c.CustomerId
    ) as  count_sales
from customers as c
```

| | FirstName | LastName | count_sales |
|---|---|---|---|
| 48 | Johannes | Van der Berg | 7 |
| 49 | Stanisław | Wójcik | 7 |
| 50 | Enrique | Muñoz | 7 |
| 51 | Joakim | Johansson | 7 |
| 52 | Emma | Jones | 7 |
| 53 | Phil | Hughes | 7 |

Коррелированный запрос, считающий продажи на каждого клиента.

```
select  FirstName,LastName,  (
    select  sum(Quantity)
    from sales_items as si
     where  si.salesid in  (
            select salesid
            from sales as s
            where  c.customerid = s.customerid
    )
) as count_track
from customers as c
```

| | FirstName | LastName | count_track |
|---|---|---|---|
| 1 | Luís | Gonçalves | 38 |
| 2 | Leonie | Köhler | 38 |
| 3 | François | Tremblay | 38 |
| 4 | Bjørn | Hansen | 32 |

Коррелированный запрос, считающий сколько треков купил каждый клиент

# СЛОЖНЫЙ ЗАПРОС НА ТРИ ТАБЛИЦЫ

В какую страну было продано самое большое количество треков в жанре джаз?

```
select ShipCountry, sum( ifnull( (
          select  sum(Quantity)
          from sales_items as si
          where   si.salesid = s.salesid
              and si.trackid  in
              (
                  select trackid
                  from  tracks as t
                  where t.genreid  in (
                      select genreid
                      from genres
                      where name ='Jazz'
                      )
          )
    ),0)) AS CNT
from sales as s
group by ShipCountry
ORDER BY  CNT DESC LIMIT  1
```

| | ShipCountry | CNT |
|---|---|---|
| 1 | USA | 22 |

Здесь несколько подзапросов и три таблицы – достаточно громоздко, и будет только хуже, если таблиц будет 10.
JOIN компактнее, но менее наглядный, сложнее проверять и доверять.

# CROSS JOIN И INNER JOIN

```
select *
from  genres  cross join  tracks  cross join  albums
```

Этот запрос равносилен запросу
Select * from genres, tracks, albums;
Это  просто декартово произведение, 31 млн. строк

```
select EmployeeId,
               employees.LastName,
               employees.FirstName,
               City,
               customers.LastName,
               customers.FirstName
from  employees cross join customers
 where employees.city=customers.city
```

| | Employee | LastName | FirstName | City | LastName | FirstName |
|---|---|---|---|---|---|---|
| 1 | 1 | Adams | Andrew | Edmonton | Philips | Mark |
| 2 | 1 | Adams | Andrew | Edmonton | Queen | Harvey |

Ищем сотрудников и клиентов, живущих в одном городе

Чаще используют INNER JOIN, у которого, в отличие от CROSS JOIN, есть условие ON.

```
select aa.name as artist,a.Title as album
from  albums as a
    inner join artists as aa
        on a.ArtistId=aa.ArtistId
```

| | artist | album |
|---|---|---|
| 1 | AC/DC | For Those About To Rock We Salute You |
| 2 | Accept | Balls to the Wall |
| 3 | Accept | Restless and Wild |
| 4 | AC/DC | Let There Be Rock |
| 5 | Aerosmith | Big Ones |
| 6 | Alanis Morissette | Jagged Little Pill |
| 7 | Alice In Chains | Facelift |
| 8 | Antônio Carlos Jobim | Warner 25 Anos |
| 9 | Apocalyptica | Plays Metallica By Four Cellos |
| 10 | Audioslave | Audioslave |
| 11 | Audioslave | Out Of Exile |
| 12 | BackBeat | BackBeat Soundtrack |
| 13 | Billy Cobham | The Best Of Billy Cobham |
| 14 | Black Label Society | Alcohol Fueled Brewtality Live! [Disc 1] |

Каждому альбому сопоставили информацию об артисте.

# INNER JOIN

```
select genres.name as  genre_name ,tracks.name tracks_name, tracks.unitprice
from  genres
    inner join tracks
     on genres.GenreId= tracks.GenreId
 where   tracks.unitprice  >2
```

| | genre_name | tracks_name | unitprice |
|---|---|---|---|
| 1 | Science Fiction | Rapture | 2.23 |
| 2 | TV Shows | Eggtown | 2.23 |
| 3 | TV Shows | Meet Kevin Johnson | 2.23 |

Показывает треки и их жанры при условии, что цена трека больше 2 у.е.

В условие ON мы обычно пишем техническое условие (id, например, по которому соединяем таблицы), а в условие WHERE аналитическое условие. В CROSS JOIN мы все пишем в WHERE, поэтому CROSS JOIN менее читабельный.

```
select FirstName, LastName, (
           select count(*)
             from sales
             where strftime('%Y', SalesDate) ='2009' and  sales.CustomerId =t.CustomerId
     ) as count_sales
from customers as t
order by count_sales desc
```

Сколько покупок сделал каждый клиент в 2009 году. Решено с помощью подзапросов.

<=>

Решено с помощью INNER JOIN.

```
select c. FirstName,c.LastName, count(s.CustomerId) as cnt_sales
from customers as c
    left join Sales as s
        on c.CustomerId =s.CustomerId and strftime('%Y', SalesDate) ='2009'
group by c.FirstName,c.LastName
order  by cnt_sales desc
```

# ЗАДАЧИ НА JOIN

1. Посчитайте общую сумму продаж в США в 1 квартале 2012 года? Решить 2-мя способами джойнами и подзапросами.
2. Покажите имена клиентов, которых нет среди работников. Решить 3-мя способами: подзапросами, джойнами и логическим вычитанием.
3. Теоретический вопрос. Вернут ли данные запросы одинаковый результат? Да или НЕТ. Объяснить почему. Какой запрос вернет больше строк ?

1

```
SELECT *  FROM t1 LEFT JOIN t2
        ON t1.column1=t2.column1
            WHERE  T1.column1=0
```

2

```
SELECT *   FROM t1 LEFT JOIN t2
        ON t1.column1=t2.column1  AND t1.column1=0
```

4. Посчитайте количество треков в каждом альбоме. В результате должно быть: имя альбома и кол-во треков. Решить 2-мя способами: подзапросом и джойнами.
5. Покажите фамилию и имя покупателей немцев сделавших заказы в 2009 году, товары которых были отгружены в город Берлин.
6. Покажите фамилии клиентов которые купили больше 30 музыкальных треков. Решить задачу 2-мя способами: джойнами и подзапросами.
7. В базе есть таблица музыкальных треков и жанров. Какова средняя стоимость музыкального трека в каждом жанре?
8. В базе есть таблица музыкальных треков и жанров. Покажите жанры, у которых средняя стоимость одного трека больше 1-го.