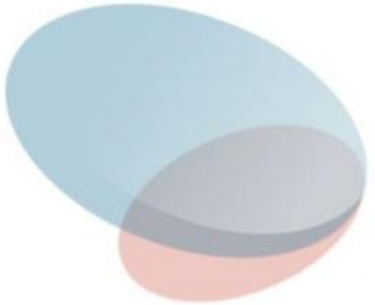


СУБД
Лекция 3
Основы языка SQL

Основы языка SQL

Пример

```
1) CREATE TABLE P(PNUM INT PRIMARY KEY,
PNAME  VARCHAR(15) NOT NULL,
PSTATUS SMALLINT(3) NOT NULL);
2) CREATE TABLE D(DNUM INT PRIMARY KEY,
DNAME  VARCHAR(15) NOT NULL,
DSTATUS SMALLINT(3) NOT NULL);
3) CREATE TABLE PD(PNUM INT NOT NULL,
DNUM INT NOT NULL,
VOLUME INT NOT NULL,
PRIMARY KEY PNUM_DNUM(PNUM, DNUM),
FOREIGN KEY (PNUM) REFERENCES P (PNUM),
FOREIGN KEY (DNUM) REFERENCES D (DNUM)
);
```

Основы языка SQL

SELECT [ALL | DISTINCT] в_выражение, ...
FROM имя_табл [син_табл], ...
[WHERE сложн_условие]
[GROUP BY полн_имя_столбца|ном_столбца, ...]
[HAVING сложн_условие]
[ORDER BY полн_имя_столбца|ном_столбца [ASC|DESC], ...];

Результатом работы оператора является выводимая на стандартный вывод вновь построенная таблица, для которой

количество и смысл столбцов определяется списком элементов **в_выражение**;

содержимое строк определяется содержимым исходных таблиц из списка **FROM** и критерием выборки, задаваемым **сложн_условие**.

Основы языка SQL

Предикат количественного сравнения ::=

Конструктор значений строки {= | < | > | <= | >= | <>}

{ANY | SOME | ALL} (Select-выражение)

Замечания. **1.** Кванторы ***ANY и SOME*** являются **синонимами** и полностью взаимозаменяемы.

2. Если указан квантор *ANY* или *SOME,* то предикат количественного сравнения возвращает *TRUE,* если сравниваемое значение **совпадает *хотя бы с одним значением***, возвращаемым в подзапросе *(select-выражении).*

3. Если указан квантор *ALL,* то предикат количественного сравнения возвращает *TRUE,* если сравниваемое значение **совпадает *с каждым значением***, возвращаемым в подзапросе.

Основы языка SQL

VI. Использование подзапросов

1) Использование предиката EXISTS. Получить список поставщиков, поставляющих деталь номер 2.

```
SELECT *
  FROM P
  WHERE EXISTS
    (SELECT *
       FROM PD
       WHERE
          PD.PNUM = P.PNUM AND
          PD.DNUM = 2);
```

Сканируется таблица поставщиков P, **каждый раз выполняется подзапрос с новым значением номера поставщика**, взятым из таблицы P.

В результат запроса включаются только те **строки** из таблицы поставщиков, для которых **вложенный подзапрос вернул непустое множество строк**.

Основы языка SQL

Замечание. В примере **вложенный подзапрос содержит параметр (внешнюю ссылку), передаваемый из основного запроса** - номер поставщика P.PNUM. **Такие подзапросы называются коррелируемыми (correlated).** Внешняя ссылка может принимать различные значения для каждой строки-кандидата, оцениваемого с помощью подзапроса, поэтому подзапрос должен выполняться заново для каждой строки, отбираемой в основном запросе. Такие подзапросы характерны для предиката EXISTS, но могут быть использованы и в других подзапросах.

Замечание. Может показаться, что запросы, содержащие коррелируемые подзапросы будут выполняться медленнее, чем запросы с некоррелируемыми подзапросами. На самом деле это не так, т.к. **то, как пользователь, сформулировал запрос, не определяет, как этот запрос будет выполняться.** Пользователь, формулирующий запрос, просто описывает, каким должен быть результат запроса, а как этот результат будет получен - за это отвечает сама СУБД.

Основы языка SQL

2) Использование предиката NOT EXISTS. Получить список поставщиков, не поставляющих деталь номер 2.

```
SELECT *
  FROM P
  WHERE NOT EXISTS
    (SELECT *
       FROM PD
       WHERE
          PD.PNUM = P.PNUM AND
          PD.DNUM = 2);
```

Замечание. Как и в предыдущем примере, здесь используется **коррелируемый подзапрос**. Отличие в том, что в основном запросе будут **отобраны** те **строки** из таблицы поставщиков, для которых **вложенный подзапрос не выдаст ни одной строки.**

Основы языка SQL

3) Получить имена поставщиков, поставляющих **все** детали.
(Получить имена поставщиков, для которых **не существуют** детали, которые они **не поставляли** бы.)

```
SELECT DISTINCT PNAME
  FROM P
  WHERE NOT EXISTS
    (SELECT *
       FROM D
       WHERE NOT EXISTS
         (SELECT *
            FROM PD
            WHERE
               PD.DNUM = D.DNUM AND
               PD.PNUM = P.PNUM));
```

Замечание. Данный **запрос содержит два вложенных подзапроса** и **реализует реляционную операцию деления отношений**.

Основы языка SQL

Самый внутренний подзапрос имеет два параметра (D.DNUM, P.PNUM), смысл подзапроса следующий: **отобрать все строки, содержащие данные о поставках поставщика с номером PNUM детали с номером DNUM.**

Отрицание NOT EXISTS означает, что **данный поставщик не поставляет данную деталь**.

Внешний к нему подзапрос, сам являющийся вложенным, имеет один параметр P.PNUM, его смысл: **отобрать список деталей, которые не поставляются поставщиком PNUM.**

Отрицание NOT EXISTS означает, что **для поставщика с номером PNUM не должно быть деталей, которые не поставлялись бы этим поставщиком**.

То есть во внешнем запросе отбираются только поставщики, поставляющие все детали.

Основы языка SQL

VII. Использование объединения, пересечения и разности

1) Объединение двух подзапросов - ключевое слово UNION. Получить имена поставщиков, имеющих статус, больший 3 **или** поставляющих хотя бы одну деталь номер 2.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  UNION
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
       PD.DNUM = 2;
```

Основы языка SQL

Замечание. Результирующие таблицы объединяемых запросов должны быть совместимы, т.е. иметь одинаковое количество столбцов и одинаковые типы столбцов в порядке их перечисления.

Не требуется, чтобы объединяемые таблицы имели бы одинаковые имена колонок. Это отличает операцию объединения запросов в SQL от операции объединения в реляционной алгебре.

Наименования колонок в результирующем запросе будут автоматически взяты из результата первого запроса в объединении.

Основы языка SQL

2) а. Пересечение двух подзапросов - ключевое слово INTERSECT. Получить имена поставщиков, имеющих статус, больший 3 **и** одновременно поставляющих хотя бы одну деталь номер 2.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  INTERSECT
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
       PD.DNUM = 2;
```

Замечание. Не все СУБД поддерживают операции пересечения и разности в операторе SELECT. Нет поддержки INTERSECT/EXCEPT, например, в MySQL, а в MS SQL Server она появилась, лишь начиная с версии 2005, и то без ключевого слова ALL. ALL с INTERSECT/EXCEPT также еще не реализована в Oracle.

Основы языка SQL

Задачу можно решить, не используя явно операцию пересечения.

2) b. Построение пересечения с помощью предиката IN и подзапроса.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
      AND P.PNAME IN
(SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2);
```

Основы языка SQL

2) с. Использование предиката существования EXISTS для построения пересечения.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  AND EXISTS (
  SELECT *
    FROM PD
    WHERE PD.PNUM = P.PNUM AND
        PD.DNUM = 2);
```

Основы языка SQL

3) а. Разность двух подзапросов - ключевое слово EXCEPT. Получить имена поставщиков, имеющих статус, больший 3, за исключением тех, кто поставляет хотя бы одну деталь номер 2.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
EXCEPT
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
       PD.DNUM = 2;
```

Основы языка SQL

Задачу можно решить, не используя явно операцию разности.

3) b. Построение разности с помощью предиката NOT IN и подзапроса.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
      AND P.PNAME NOT IN
(SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2);
```

Основы языка SQL

3) с. Использование предиката существования NOT EXISTS для построения разности.

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
AND NOT EXISTS (
  SELECT *
    FROM PD
    WHERE PD.PNUM = P.PNUM AND
        PD.DNUM = 2);
```

Основы языка SQL

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

Для того, чтобы показать, что язык SQL является реляционно полным, нужно показать, что **любой реляционный оператор может быть выражен средствами SQL**. На самом деле **достаточно показать**, что средствами SQL можно выразить **любой из примитивных реляционных операторов**.

1. Оператор декартового произведения

Реляционная алгебра: *A* TIMES *B.*

Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A, B;
```

или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A CROSS JOIN B;
```

Основы языка SQL

2. Оператор проекции

Реляционная алгебра: *A*[*X*, *Y*, …,*Z*].

Оператор SQL:

```
SELECT DISTINCT X, Y, …, Z
    FROM A;
```

3. Оператор выборки

Реляционная алгебра: A WHERE c .

Оператор SQL:

```
SELECT *
    FROM A
    WHERE c;
```

Основы языка SQL

4. Оператор объединения

Реляционная алгебра: *A* UNION *B*.

Оператор SQL:

```
SELECT *
    FROM A
    UNION
    SELECT *
        FROM B;
```

5. Оператор вычитания

Реляционная алгебра: *A* MINUS *B.*

Оператор SQL:

```
SELECT *
    FROM A
    EXCEPT
    SELECT *
        FROM B;
```

```
SELECT A.X
    FROM A
    WHERE A.X NOT IN
(SELECT B.X
        FROM B);
```

Основы языка SQL

Реляционный оператор переименования RENAME выражается при помощи **ключевого слова AS в списке отбираемых полей** оператора SELECT.

Таким образом, язык SQL является реляционно полным.

Остальные операторы реляционной алгебры (соединение, пересечение, деление) выражаются через примитивные, следовательно, могут быть выражены операторами SQL. Приведем их для практического применения.

Основы языка SQL

6. Оператор соединения

Реляционная алгебра: (*A* TIMES *B*) WHERE *c.*

Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A, B
    WHERE c;
```

или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A CROSS JOIN B
    WHERE c;
```

Основы языка SQL

7. Оператор пересечения

Реляционная алгебра: *A* INTERSECT *B.*

Оператор SQL:

```
SELECT *                SELECT A.X
   FROM A                  FROM A
   INTERSECT               WHERE A.X IN
   SELECT *               (SELECT B.X
      FROM B;                    FROM B);
```

Основы языка SQL

8. Оператор деления

Реляционная алгебра: *A* DIVIDEBY *B.*

Оператор SQL:

```
SELECT DISTINCT A.X
  FROM A
  WHERE NOT EXIST
    (SELECT *
       FROM B
       WHERE NOT EXIST
         (SELECT *
            FROM A A1
            WHERE
               A1.X = A.X AND
               A1.Y = B.Y));
```

Основы языка SQL

Замечание. Дадим пример эквивалентного преобразования выражений, представляющих суть запроса.

Пусть отношение **A** содержит данные о **поставках** деталей, отношение **B** содержит **список всех деталей**, которые могут поставляться. Атрибут **X** является **номером поставщика**, атрибут **Y** является **номером детали**.

Разделить отношение A на отношение B означает в данном примере "**отобрать номера поставщиков, которые поставляют все детали**".

1. "Отобрать номера поставщиков, которые поставляют все детали" эквивалентно
2. "Отобрать те номера поставщиков из таблицы A, для которых не существует не поставляемых деталей в таблице B" эквивалентно

Основы языка SQL

3. "Отобрать те номера поставщиков из таблицы A, для которых не существует тех номеров деталей из таблицы B, которые не поставляются этим поставщиком" эквивалентно

4. "Отобрать те номера поставщиков из таблицы A, для которых не существует тех номеров деталей из таблицы B, для которых не существует записей о поставках в таблице A для этого поставщика и этой детали".

Последнее выражение дословно переводится на язык SQL.

При переводе выражения на язык SQL во внутреннем подзапросе таблица A должна быть переименована, для того чтобы отличать ее от экземпляра этой же таблицы, используемой во внешнем запросе.

Основы языка SQL

Запросы, невыразимые средствами реляционной алгебры. Транзитивное замыкание

Определение. Пусть отношение R задано на декартовом квадрате A^2 некоторого множества. Транзитивным замыканием R называется новое отношение R1, состоящее из кортежей (x, y), для которых выполняется:

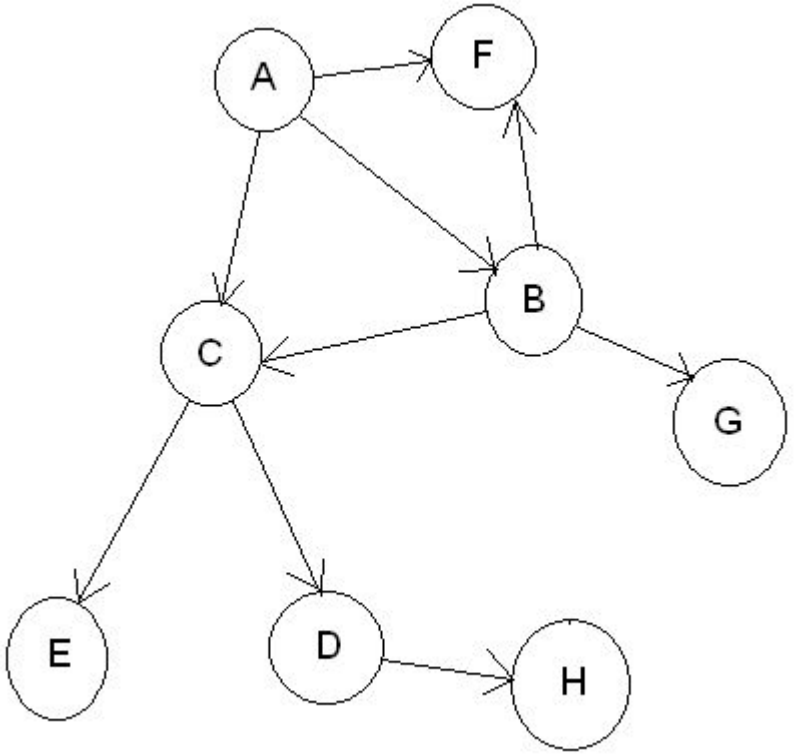
- либо кортеж $(x, y) \in R$,
- либо найдется конечная последовательность элементов $z_1, z_2, \ldots, z_n \in A$ такая, что все кортежи $(x,z_1), (z_1,z_2), \ldots, (z_n, y) \in R$.

Очевидно, $R \subseteq R1$.

Основы языка SQL

Пример. Задан граф и таблица достижимости пунктов.

1. Построить таблицу достижимости пунктов в точности через один транзитный пункт.
2. Построить таблицу достижимости пунктов в точности через два транзитных пункта.
3. Построить транзитивное замыкание.

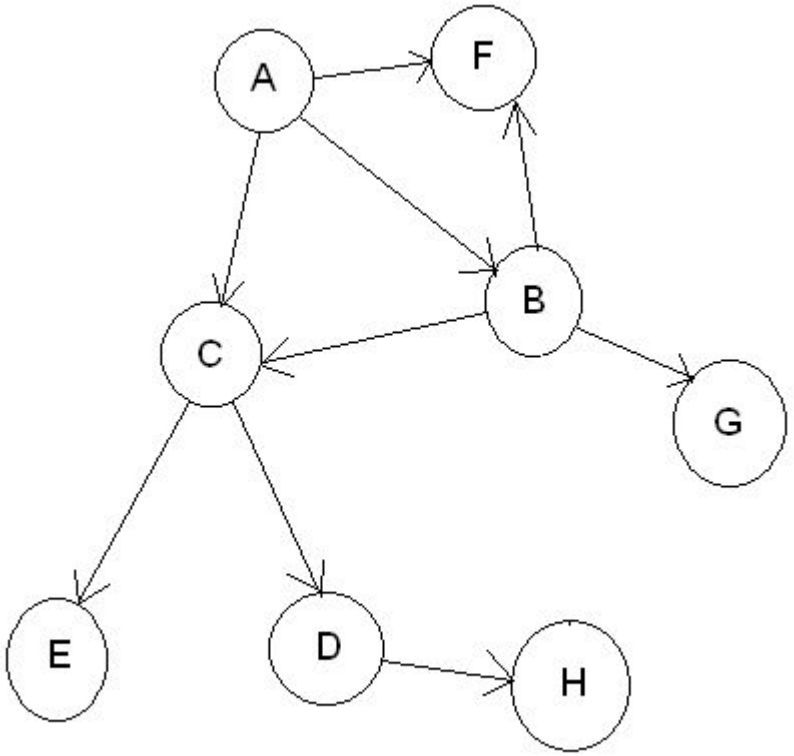


starting_point	terminal_point
A	B
A	C
A	F
B	C
B	G
B	F
C	E
C	D
D	H

Основы языка SQL

1. SELECT F1.starting_point, F2.terminal_point FROM flight F1, flight F2
WHERE F1.terminal_point=F2.starting_point;

2. SELECT F1.starting_point, F3.terminal_point FROM flight F1, flight F2, flight F3
WHERE F1.terminal_point=F2.starting_point AND F2.terminal_point=F3.starting_point;

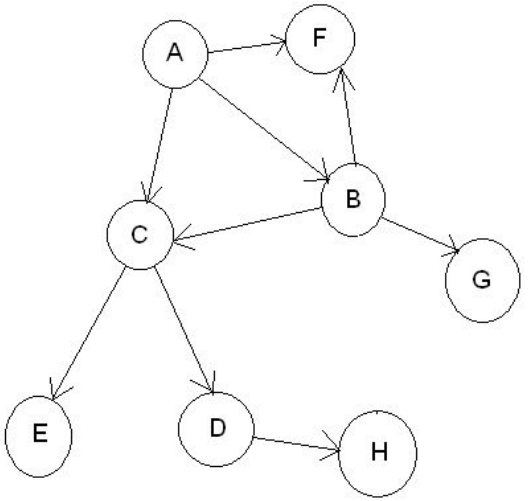


starting_point	terminal_point
A	C
A	G
A	F
A	E
B	E
A	D
B	D
C	H

starting_point	terminal_point
A	E
A	D
A	H
B	H

Основы языка SQL

3. SELECT * FROM flight
UNION
SELECT F1.starting_point, F2.terminal_point
FROM flight F1, flight F2
WHERE F1.terminal_point=F2.starting_point
UNION
SELECT F1.starting_point, F3.terminal_point
FROM flight F1, flight F2, flight F3
WHERE F1.terminal_point=F2.starting_point
AND F2.terminal_point=F3.starting_point
ORDER BY 1;



starting_point	terminal_point
A	B
A	C
A	F
A	G
A	E
A	D
A	H
B	C
B	G
B	F
B	E
B	D
B	H
C	E
C	D
C	H
D	H

Основы языка SQL

Синтаксис оператора выборки данных (SELECT)

При описании синтаксиса операторов обычно используются условные обозначения, известные как **стандартные формы Бэкуса-Наура (BNF).**

- Символ "::=" означает равенство по определению.
- Необязательные элементы оператора заключены в квадратные скобки [].
- Вертикальная черта | указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты.
- Фигурные скобки {} указывают на то, что все находящееся внутри них является единым целым.
- Троеточие "…" означает, что предшествующая часть оператора может быть повторена любое количество раз.

Основы языка SQL

Синтаксис оператора выборки в упрощенном виде

Оператор выборки **::=**

Табличное выражение

[ORDER BY

{{*Имя столбца-результата* [ASC | DESC]} |
{*Положительное целое* [ASC | DESC]}.,..];

Табличное выражение **::=**

Select-выражение

[{UNION | INTERSECT | EXCEPT} [ALL]

{*Select-выражение* | TABLE *Имя таблицы* | *Конструктор значений таблицы*}]

Основы языка SQL

Select-выражение **::=**

SELECT [ALL | DISTINCT]

{{{*Скалярное выражение* | *Функция агрегирования* | *Select-выражение*} [AS *Имя столбца*]}.,..}

| {{*Имя таблицы* | *Имя корреляции*}.*} | *

FROM {

{*Имя таблицы* [AS] [*Имя корреляции*] [(*Имя столбца*.,..)]}

| {*Select-выражение* [AS] *Имя корреляции* [(*Имя столбца*.,..)]}

| *Соединенная таблица* }.,..

[WHERE *Условное выражение*]

[GROUP BY {[{*Имя таблицы* | *Имя корреляции*}.] *Имя столбца*}.,..]

[HAVING *Условное выражение*]

Основы языка SQL

Скалярное выражение - в качестве скалярных выражений в разделе SELECT могут выступать либо имена столбцов таблиц, входящих в раздел FROM, либо простые функции, возвращающие скалярные значения.

Функция агрегирования **::=** COUNT (*) |

{

{COUNT | MAX | MIN | SUM | AVG} ([ALL | DISTINCT] *Скалярное выражение*)

}

Конструктор значений таблицы **::=**

VALUES *Конструктор значений строки*.,..

Конструктор значений строки **::=**

Элемент конструктора | (*Элемент конструктора*.,..) | *Select-выражение*

Замечание. *Select-выражение*, используемое в конструкторе значений строки, обязано возвращать ровно одну строку.

Элемент конструктора **::=**

Выражение для вычисления значения | NULL | DEFAULT

Основы языка SQL

Синтаксис соединенных таблиц

В разделе FROM оператора SELECT можно использовать **соединенные таблицы**. Пусть в результате некоторых операций мы получаем таблицы A и B. Такими операциями могут быть, например, оператор SELECT или другая соединенная таблица. Тогда синтаксис соединенной таблицы имеет следующий вид:

Соединенная таблица **::=**

Перекрестное соединение

| *Естественное соединение*

| *Соединение посредством предиката*

| *Соединение посредством имен столбцов*

| *Соединение объединения*

Основы языка SQL

Тип соединения ::= *INNER | LEFT [OUTER]*
| RIGTH [OUTER]
| FULL [OUTER]

Перекрестное соединение ::=

Таблица A CROSS JOIN Таблица B

Естественное соединение ::=

Таблица A [NATURAL] [Тип соединения] JOIN Таблица B

Соединение посредством предиката ::=

Таблица A [Тип соединения] JOIN Таблица B ON
Предикат

Соединение посредством имен столбцов ::=

Таблица A [Тип соединения] JOIN Таблица B USING
(Имя столбца.,..)

Соединение объединения ::=

Таблица A UNION JOIN Таблица B

Основы языка SQL

CROSS JOIN - Перекрестное соединение возвращает просто **декартово произведение** таблиц. Такое соединение в разделе FROM может быть заменено списком таблиц через запятую.

NATURAL JOIN - **естественное соединение** производится по всем столбцам таблиц A и B, имеющим одинаковые имена. В результатирующую таблицу одинаковые столбцы вставляются только один раз.

JOIN … ON - соединение посредством предиката соединяет строки таблиц A и B посредством указанного предиката.

JOIN … USING - соединение посредством имен столбцов соединяет отношения подобно естественному соединению по тем общим столбцам таблиц А и Б, которые указаны в списке USING.

Основы языка SQL

OUTER - ключевое слово OUTER (внешний) не является обязательным, оно **не используется** ни в каких операциях с данными.

INNER - тип соединения "внутреннее". Внутренний тип соединения **используется по умолчанию**, когда тип явно не задан. В таблицах A и B **соединяются только те строки, для которых найдено совпадение**.

LEFT (OUTER) - тип соединения "левое (внешнее)". Левое соединение таблиц A и B **включает в себя все строки из левой таблицы A** и **те строки из правой таблицы B, для которых обнаружено совпадение**. Для строк из таблицы A, для которых не найдено соответствия в таблице B, в столбцы, извлекаемые из таблицы B, заносятся значения NULL.

Основы языка SQL

RIGHT (OUTER) - тип соединения "правое (внешнее)". Правое соединение таблиц A и B включает в себя **все строки из правой таблицы B и те строки из левой таблицы A, для которых обнаружено совпадение**. Для строк из таблицы B, для которых не найдено соответствия в таблице A, в столбцы, извлекаемые из таблицы A заносятся значения NULL.

FULL (OUTER) - тип соединения "полное (внешнее)". Это **комбинация левого и правого соединений**. В полное соединение включаются все строки из обеих таблиц. Для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений.

Основы языка SQL

UNION JOIN - **соединение объединения** является обратным по отношению к внутреннему соединению. Оно **включает только те строки из таблиц A и B, для которых не найдено совпадений**. В них используются **значения NULL для столбцов, полученных из другой таблицы**.

Если взять полное внешнее соединение и удалить из него строки, полученные в результате внутреннего соединения, то получится соединение объединения.

Основы языка SQL

Примеры

1. Получить список сотрудников и названия их отделов, включая сотрудников, еще не назначенных ни в какой отдел.

```
SELECT first_name as ‘имя’, last_name as ‘фамилия’,
    patronimic as ‘отчество’, department as ‘отдел’
 FROM employee e LEFT JOIN department d
 ON e.dept_no = d.dept_no;
```

2. Получить список сотрудников и названия их отделов, включая отделы, в которые еще не назначены сотрудники.

```
SELECT first_name, last_name, department
 FROM employee e RIGHT JOIN department d
 ON e.dept_no = d.dept_no;
```

Основы языка SQL

3. Получить список сотрудников и названия отделов, включая сотрудников, еще не назначенных ни в какой отдел, и отделы, в которые еще не назначены сотрудники (не используется в MySQL).

```
SELECT first_name, last_name, department
 FROM employee e FULL JOIN department d
 ON e.dept_no = d.dept_no;
```

4. Получить список сотрудников, еще не назначенных ни в какой отдел, и названия отделов, в которые еще не назначены сотрудники (не используется в MySQL).

```
SELECT first_name, last_name, department
 FROM employee e UNION JOIN department d
 ON e.dept_no = d.dept_no;
```

Основы языка SQL

Синтаксис условных выражений раздела WHERE

Условное выражение **::=**

[( ] [NOT]

{*Предикат сравнения*

| *Предикат between*

| *Предикат in*

| *Предикат like*

| *Предикат null*

| *Предикат количественного сравнения*

| *Предикат exists*

| *Предикат unique*

| *Предикат match*

| *Предикат overlaps*}

[{AND | OR} *Условное выражение*] [ )]

[IS [NOT] {TRUE | FALSE | UNKNOWN}]

Основы языка SQL

***Предикат сравнения* ::=**

Конструктор значений строки {= | < | > | <= | >= | <>}
Конструктор значений строки

Предикат between ::=

Конструктор значений строки [NOT] BETWEEN
Конструктор значений строки AND Конструктор значений строки

Предикат in ::=

Конструктор значений строки [NOT] IN
{(Select-выражение) | (Выражение для вычисления значения.,..)}

Предикат like ::=

Выражение для вычисления значения строки-поиска [NOT] LIKE
Выражение для вычисления значения строки-шаблона [ESCAPE Символ]

Основы языка SQL

Предикат null ::=

Конструктор значений строки IS [NOT] NULL

Предикат количественного сравнения ::=

Конструктор значений строки {= | < | > | <= | >= | <>}
{ANY | SOME | ALL} (Select-выражение)

Замечания. **1.** Кванторы ***ANY и SOME*** являются **синонимами** и полностью взаимозаменяемы.

2. Если указан квантор *ANY* или *SOME,* то предикат количественного сравнения возвращает *TRUE,* если сравниваемое значение **совпадает *хотя бы с одним значением***, возвращаемым в подзапросе *(select-выражении).*

3. Если указан квантор *ALL,* то предикат количественного сравнения возвращает *TRUE,* если сравниваемое значение **совпадает *с каждым значением***, возвращаемым в подзапросе.

Основы языка SQL

Пример

P.PNUM = SOME (SELECT PD.PNUM FROM PD WHERE PD.DNUM=2)

***Предикат exists* ::=**

EXISTS (*Select-выражение*)

Замечание. Предикат EXISTS возвращает значение TRUE, если **результат подзапроса** (select-выражения) **не пуст**.

***Предикат unique* ::=**

UNIQUE (*Select-выражение*)

Замечание. Предикат UNIQUE возвращает TRUE, если **в результате подзапроса** (select-выражения) **нет совпадающих строк**.

Основы языка SQL

***Предикат match* ::=**

Конструктор значений строки MATCH [UNIQUE]

[PARTIAL | FULL] (*Select-выражение*)

Замечание. Предикат MATCH **проверяет, будет ли значение**, определенное в конструкторе строки **совпадать со значением любой строки,** полученной в результате подзапроса.

***Предикат overlaps* ::=**

Конструктор значений строки OVERLAPS *Конструктор значений строки*

Замечание. Предикат OVERLAPS, является специализированным предикатом, позволяющем **определить, будет ли указанный период времени перекрывать другой период времени**.

Основы языка SQL

Порядок выполнения оператора SELECT

Различают **концептуальную схему выполнения оператора SELECT** и **фактическую схему его выполнения**.

Концептуальная схема описывает, в какой **логической последовательности** должны выполняться операции**, чтобы получить результат**.

При реальном (фактическом) выполнении оператора SELECT на первый план выступает достижение максимальной скорости выполнения запроса. Для этого используется **оптимизатор**, который, анализируя различные планы выполнения запроса, выбирает наилучший из них.

Основы языка SQL

Концептуальная схема выполнения оператора SELECT

Стадия 1. Выполнение **одиночного** оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов.

Шаг 1 (FROM). Вычисляется прямое **декартовое произведение** всех таблиц, указанных в обязательном разделе FROM. В результате шага 1 получаем таблицу A.

Шаг 2 (WHERE). Если в операторе SELECT присутствует раздел WHERE, то для каждой строки из таблицы A, полученной при выполнении шага 1, вычисляется условное выражение, приведенное в разделе WHERE. **Те строки, для которых условное выражение возвращает значение TRUE, включаются в результат**.

Если раздел WHERE опущен, то сразу переходим к шагу 3.

Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой.

В результате шага 2 получаем таблицу B.

Основы языка SQL

Шаг 3 (GROUP BY). Если в операторе SELECT присутствует раздел GROUP BY, то **строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки**, приведенным в разделе GROUP BY.

Если раздел GROUP BY опущен, то сразу переходим к шагу 4.

В результате шага 3 получаем таблицу C.

Шаг 4 (HAVING). Если в операторе SELECT присутствует раздел HAVING, то **группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются**.

Если раздел HAVING опущен, то сразу переходим к шагу 5.

В результате шага 4 получаем таблицу D.

Основы языка SQL

Шаг 5 (SELECT). Каждая группа, полученная на шаге 4, **генерирует одну строку результата** следующим образом.

Вычисляются все скалярные выражения, указанные в разделе SELECT. По правилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы.

Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT.

Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что **имеется всего одна группа.**

Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что **имеется столько групп, сколько строк отобрано к данному моменту**.

В результате шага 5 получаем таблицу E, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.

Стадия 3. Упорядочение результата

Если в операторе SELECT присутствует раздел ORDER BY, то **строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения**, приведенным в разделе ORDER BY.

Основы языка SQL

Фактическая схема выполнения оператора SELECT

Выполнять приведенный концептуальный алгоритм непосредственно чрезвычайно неэкономно. Даже когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.

В РСУБД имеется **оптимизатор**, функцией которого является **нахождение такого оптимального алгоритма выполнения запроса, который гарантирует получение правильного результата**.

Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов.

Основы языка SQL

Схема работы оптимизатора

Шаг 1 (Синтаксический анализ). Поступивший запрос подвергается синтаксическому анализу.

На этом шаге определяется, **правильно ли** вообще (с точки зрения синтаксиса SQL) **сформулирован запрос**.

В ходе синтаксического анализа вырабатывается некоторое **внутренне представление запроса**, используемое на последующих шагах.

Основы языка SQL

Шаг 2 (Преобразование в каноническую форму). При преобразовании запроса во внутреннем представлении к канонической форме используются как синтаксические, так и семантические преобразования.

Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить **новое внутренне представление запроса, синтаксически эквивалентное исходному**, но стандартное в некотором смысле.

Семантические преобразования используют дополнительные знания, которыми владеет система, например, **ограничения целостности**. В результате семантических преобразований получается **запрос, синтаксически не эквивалентный исходному, но дающий тот же самый результат.**

Основы языка SQL

Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана). На этом шаге оптимизатор генерирует **множество возможных планов выполнения запроса**.

Каждый план строится как **комбинация низкоуровневых процедур** доступа к данным таблиц и методам соединения таблиц.

Из всех сгенерированных планов **выбирается план, обладающий минимальной стоимостью**.

При этом **анализируются** данные о **наличии индексов** у таблиц, сведения о наличии **статистических данных о распределении значений в таблицах**, и т.п.

Стоимость плана это, как правило, **сумма стоимостей выполнения отдельных низкоуровневых процедур**, которые используются для его выполнения.

В **стоимость** выполнения **отдельной процедуры** могут входить **оценки количества обращений к дискам**, **степень загруженности процессора** и другие параметры.

Шаг 4. (Выполнение плана запроса). План, выбранный на предыдущем шаге, **передается на реальное выполнение**.

Во многом качество конкретной СУБД определяется качеством ее оптимизатора.

Хороший оптимизатор может повысить скорость выполнения запроса на несколько порядков.

Качество оптимизатора определяется тем,

- какие **методы преобразований** он может использовать,
- какой **статистической и иной информацией** о таблицах он располагает,
- какие **методы для оценки стоимости выполнения плана** он знает.