Введение в системы управления базами данных
А.Ю.Пушников

Данное учебное пособие было опубликовано в 1999 г. в двух частях издательством Башкирского государственного университета. Выходные данные: УДК 519.6, ББК 32.973-018.2 Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с. - ISBN 5-7477-0350-1. В электронной версии - полный текст книги

Web-страница: http://www.citforum.ru/database/dblearn/index.shtml

ЭЛЕМЕНТЫ ЯЗЫКА SQL

В данной главе рассматриваются элементы языка SQL (Structured Query Language). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже (SQL:2008).

Язык SQL стал фактически стандартным языком доступа к базам данных. Все СУБД, претендующие на название "реляционные", реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют в настоящее время средства доступа к реляционным данным. Целью стандартизации является переносимость приложений между различными СУБД.

Нужно заметить, что в настоящее время, ни одна система не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект - это надмножество некоторого подмножества стандарта SQL. Это затрудняет переносимость приложений, разработанных для одних СУБД в другие СУБД.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее. Например, отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия.

Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

Операторы SQL

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям.

Можно выделить следующие группы операторов (перечислены не все операторы SQL):

Операторы DDL (Data Definition Language) - операторы определения объектов базы данных

- CREATE SCHEMA - создать схему базы данных
- DROP SHEMA - удалить схему базы данных
- CREATE TABLE - создать таблицу
- ALTER TABLE - изменить таблицу
- DROP TABLE - удалить таблицу
- CREATE DOMAIN - создать домен
- ALTER DOMAIN - изменить домен
- DROP DOMAIN - удалить домен
- CREATE COLLATION - создать последовательность
- DROP COLLATION - удалить последовательность
- CREATE VIEW - создать представление
- DROP VIEW - удалить представление

Операторы DML (Data Manipulation Language) - операторы манипулирования данными

- SELECT - отобрать строки из таблиц
- INSERT - добавить строки в таблицу
- UPDATE - изменить строки в таблице
- DELETE - удалить строки в таблице
- COMMIT - зафиксировать внесенные изменения
- ROLLBACK - откатить внесенные изменения

Операторы защиты и управления данными

- CREATE ASSERTION - создать ограничение
- DROP ASSERTION - удалить ограничение
- GRANT - предоставить привилегии пользователю или приложению на манипулирование объектами
- REVOKE - отменить привилегии пользователя или приложения

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Наиболее важными для пользователя являются операторы манипулирования данными (DML).

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

Пример 1. Вставка одной строки в таблицу:

```
INSERT INTO
  P (PNUM, PNAME)
  VALUES (4, "Иванов");
```

Пример 2. Вставка в таблицу нескольких строк, выбранных из другой таблицы (в таблицу TMP_TABLE вставляются данные о поставщиках из таблицы P, имеющие номера, большие 2):

```
INSERT INTO
  TMP_TABLE (PNUM, PNAME)
  SELECT PNUM, PNAME
    FROM P
    WHERE P.PNUM>2;
```

UPDATE - обновление строк в таблице

Пример 3. Обновление нескольких строк в таблице:

```
UPDATE P
  SET PNAME = "Пушников"
  WHERE P.PNUM = 1;
```

DELETE - удаление строк в таблице

Пример 4. Удаление нескольких строк в таблице:

```
DELETE FROM P
  WHERE P.PNUM = 1;
```

Пример 5. Удаление всех строк в таблице:

```
DELETE FROM P;
```

Примеры использования оператора SELECT

Оператор SELECT является фактически самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных - предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных.

Замечание. На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления. Представления - это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления - это таблица, которая не хранится постоянно в базе данных, а "возникает" в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между хранимыми таблицами и представлениями, но эти различия *скрыты* от пользователя.

Результатом выполнения оператора SELECT всегда является таблица. Таким образом, по результатам действий оператор SELECT похож на операторы реляционной алгебры. Любой оператор реляционной алгебры может быть выражен подходящим образом сформулированным оператором SELECT. Сложность оператора SELECT определяется тем, что он содержит в себе все возможности реляционной алгебры, а также дополнительные возможности, которых в реляционной алгебре нет.

Отбор данных из одной таблицы

Пример 6. Выбрать все данные из таблицы поставщиков (ключевые слова ***SELECT***… ***FROM***…):

```
SELECT *
  FROM P;
```

Замечание. В результате получим новую таблицу, содержащую полную копию данных из исходной таблицы P.

Пример 7. Выбрать все строки из таблицы поставщиков, удовлетворяющих некоторому условию (ключевое слово ***WHERE***…):

```
SELECT *
  FROM P
  WHERE P.PNUM > 2;
```

Замечание. В качестве условия в разделе WHERE можно использовать сложные логические выражения, использующие поля таблиц, константы, сравнения (>, <, = и т.д.), скобки, союзы AND и OR, отрицание NOT.

Пример 8. Выбрать некоторые колонки из исходной таблицы (указание списка отбираемых колонок):

```
SELECT P.NAME
  FROM P;
```

Замечание. В результате получим таблицу с одной колонкой, содержащую все наименования поставщиков.

Замечание. Если в исходной таблице присутствовало несколько поставщиков с разными номерами, но одинаковыми наименованиями, то в результирующей таблице *будут строки с повторениями* - дубликаты строк автоматически не отбрасываются.

Пример 9. Выбрать некоторые колонки из исходной таблицы, удалив из результата повторяющиеся строки (ключевое слово ***DISTINCT***):

```
SELECT DISTINCT P.NAME
  FROM P;
```

Замечание. Использование ключевого слова DISTINCT приводит к тому, что в результирующей таблице будут удалены все повторяющиеся строки.

Пример 10. Использование скалярных выражений и переименований колонок в запросах (ключевое слово ***AS***…):

```
SELECT
    TOVAR.TNAME,
    TOVAR.KOL,
    TOVAR.PRICE,
    "=" AS EQU,
    TOVAR.KOL*TOVAR.PRICE AS SUMMA
  FROM TOVAR;
```

В результате получим таблицу с колонками, которых не было в исходной таблице TOVAR:

TNAME	KOL	PRICE	EQU	SUMMA
Болт	10	100	=	1000
Гайка	20	200	=	4000
Винт	30	300	=	9000

Пример 11.Упорядочение результатов запроса (ключевое слово ***ORDER BY***…):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY DNUM;
```

В результате получим следующую таблицу, упорядоченную по полю DNUM:

PNUM	DNUM	VOLUME
1	1	100
2	1	150
3	1	1000
1	2	200
2	2	250
1	3	300

Пример 12. Упорядочение результатов запроса по нескольким полям с возрастанием или убыванием (ключевые слова ***ASC***, ***DESC***):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY
    DNUM ASC,
    VOLUME DESC;
```

В результате получим таблицу, в которой строки идут в порядке возрастания значения поля DNUM, а строки, с одинаковым значением DNUM идут в порядке убывания значения поля VOLUME:

PNUM	DNUM	VOLUME
3	1	1000
2	1	150
1	1	100
2	2	250
1	2	200
1	3	300

Замечание. Если явно не указаны ключевые слова ASC или DESC, то по умолчанию принимается упорядочение по возрастанию (ASC).

Отбор данных из нескольких таблиц

Пример 13. Естественное соединение таблиц (способ 1 - явное указание условий соединения):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P, PD
  WHERE P.PNUM = PD.PNUM;
```

В результате получим новую таблицу, в которой строки с данными о поставщиках соединены со строками с данными о поставках деталей:

PNUM	PNAME	DNUM	VOLUME
1	Иванов	1	100
1	Иванов	2	200
1	Иванов	3	300
2	Петров	1	150
2	Петров	2	250
3	Сидоров	1	1000

Замечание. Соединяемые таблицы перечислены в разделе FROM оператора, условие соединения приведено в разделе WHERE. Раздел WHERE, помимо условия соединения таблиц, может также содержать и условия отбора строк.

Пример 14. Естественное соединение таблиц (способ 2 - ключевые слова ***JOIN… USING…***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P JOIN PD USING PNUM;
```

Замечание. Ключевое слово USING позволяет *явно указать*, по каким из *общих* колонок таблиц будет производиться соединение.

Пример 15. Естественное соединение таблиц (способ 3 - ключевое слово ***NATURAL JOIN***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P NATURAL JOIN PD;
```

Замечание. В разделе FROM не указано, по каким полям производится соединение. NATURAL JOIN автоматически соединяет *по всем одинаковым полям* в таблицах.

Пример 16. Естественное соединение трех таблиц:

```
SELECT
    P.PNAME,
    D.DNAME,
    PD.VOLUME
  FROM
    P NATURAL JOIN PD NATURAL JOIN D;
```

В результате получим следующую таблицу:

PNAME	DNAME	VOLUME
Иванов	Болт	100
Иванов	Гайка	200
Иванов	Винт	300
Петров	Болт	150
Петров	Гайка	250
Сидоров	Болт	1000

Пример 17. Прямое произведение таблиц:

```
SELECT
    P.PNUM,
    P.PNAME,
    D.DNUM,
    D.DNAME
  FROM P, D;
```

В результате получим следующую таблицу:

PNUM	PNAME	DNUM	DNAME
1	Иванов	1	Болт
1	Иванов	2	Гайка
1	Иванов	3	Винт
2	Петров	1	Болт
2	Петров	2	Гайка
2	Петров	3	Винт
3	Сидоров	1	Болт
3	Сидоров	2	Гайка
3	Сидоров	3	Винт

Замечание. Т.к. не указано условие соединения таблиц, то *каждая* строка первой таблицы соединится с *каждой* строкой второй таблицы.

Пример 18. Соединение таблиц по произвольному условию. Рассмотрим таблицы поставщиков и деталей, которыми присвоен некоторый статую (см. пример 8 из предыдущей главы):

PNUM	PNAME	PSTATUS
1	Иванов	4
2	Петров	1
3	Сидоров	2

Таблица 1 Отношение P (Поставщики)

DNUM	DNAME	DSTATUS
1	Болт	3
2	Гайка	2
3	Винт	1

Таблица 2 Отношение D (Детали)

Ответ на вопрос "какие поставщики имеют право поставлять какие детали?" дает следующий запрос:

```
SELECT
    P.PNUM,
    P.PNAME,
    P.PSTATUS,
    D.DNUM,
    D.DNAME,
    D.DSTATUS
  FROM P, D
  WHERE P.PSTATUS >= D.DSTATUS;
```

В результате получим следующую таблицу:

PNUM	PNAME	PSTATUS	DNUM	DNAME	DSTATUS
1	Иванов	4	1	Болт	3
1	Иванов	4	2	Гайка	2
1	Иванов	4	3	Винт	1
2	Петров	1	3	Винт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Винт	1

Использование имен корреляции (алиасов, псевдонимов)

Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются ***имена корреляции*** (***алиасы***, ***псевдонимы***), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени таблицы. Имена корреляции должны использоваться в качестве префикса перед именем

столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

Пример 19. Отобрать все пары поставщиков таким образом, чтобы первый поставщик в паре имел статус, больший статуса второго поставщика:

```
SELECT
    P1.PNAME AS PNAME1,
    P1.PSTATUS AS PSTATUS1,
    P2.PNAME AS PNAME2,
    P2.PSTATUS AS PSTATUS2
  FROM
    P P1, P P2
  WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

В результате получим следующую таблицу:

PNAME1	PSTATUS1	PNAME2	PSTATUS2
Иванов	4	Петров	1
Иванов	4	Сидоров	2
Сидоров	2	Петров	1

Пример 20. Рассмотрим ситуацию, когда некоторые поставщики (назовем их контрагенты) могут выступать как в качестве поставщиков деталей, так и в качестве получателей. Таблицы, хранящие данные могут иметь следующий вид:

Номер контрагента NUM	Наименование контрагента NAME
1	Иванов
2	Петров
3	Сидоров

Таблица 3 Отношение CONTRAGENTS

Номер детали DNUM	Наименование детали DNAME
1	Болт
2	Гайка
3	Винт

Таблица 4 Отношение DETAILS (Детали)

Номер поставщика PNUM	Номер получателя CNUM	Номер детали DNUM	Поставляемое количество VOLUME
1	2	1	100
1	3	2	200
1	3	3	300
2	3	1	150
2	3	2	250
3	1	1	1000

Таблица 5 Отношение CD (Поставки)

В таблице CD (поставки) поля PNUM и CNUM являются внешними ключами, ссылающимися на потенциальный ключ NUM в таблице CONTRAGENTS.

Ответ на вопрос "кто кому что в каком количестве поставляет" дается следующим запросом:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS,
    CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM AND
    D.DNUM = CD.DNUM;
```

В результате получим следующую таблицу:

Наименование поставщика PNAME	Наименование получателя CNAME	Наименование детали DNAME	Поставляемое количество VOLUME
Иванов	Петров	Болт	100
Иванов	Сидоров	Гайка	200
Иванов	Сидоров	Винт	300
Петров	Сидоров	Болт	150
Петров	Сидоров	Гайка	250
Сидоров	Иванов	Болт	1000

Замечание. Этот же запрос может быть выражен очень большим количеством способов, например, так:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS NATURAL JOIN CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM;
```

Использование агрегатных функций в запросах

Пример 21. Получить общее количество поставщиков (ключевое слово ***COUNT***):

```
SELECT COUNT(*) AS N
  FROM P;
```

В результате получим таблицу с одним столбцом и одной строкой, содержащей количество строк из таблицы P:

N
3

Пример 22. Получить общее, максимальное, минимальное и среднее количества поставляемых деталей (ключевые слова ***SUM***, ***MAX***, ***MIN***, ***AVG***):

```
SELECT
    SUM(PD.VOLUME) AS SM,
    MAX(PD.VOLUME) AS MX,
    MIN(PD.VOLUME) AS MN,
    AVG(PD.VOLUME) AS AV
  FROM PD;
```

В результате получим следующую таблицу с одной строкой:

SM	MX	MN	AV
2000	1000	100	333.33333333

Использование агрегатных функций с группировками

Пример 23. Для каждой детали получить суммарное поставляемое количество (ключевое слово ***GROUP BY***…):

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Этот запрос будет выполняться следующим образом. Сначала строки исходной таблицы будут сгруппированы так, чтобы в каждую группу попали строки с одинаковыми значениями DNUM. Потом внутри каждой группы будет просуммировано поле VOLUME. От каждой группы в результатирующую таблицу будет включена одна строка:

DNUM	SM
1	1250
2	450
3	300

Замечание. В списке отбираемых полей оператора SELECT, содержащего раздел GROUP BY можно включать *только* агрегатные функции и поля, *которые входят в условие группировки*. Следующий запрос выдаст синтаксическую ошибку:

```
SELECT
    PD.PNUM,
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Причина ошибки в том, что в список отбираемых полей включено поле PNUM, которое *не входит* в раздел GROUP BY. И действительно, в каждую полученную группу строк может входить несколько строк с *различными* значениями поля PNUM. Из каждой группы строк будет сформировано по одной итоговой строке. При этом нет однозначного ответа на вопрос, какое значение выбрать для поля PNUM в итоговой строке.

Замечание. Некоторые диалекты SQL не считают это за ошибку. Запрос будет выполнен, но предсказать, какие значения будут внесены в поле PNUM в результатирующей таблице, невозможно.

Пример 24. Получить номера деталей, суммарное поставляемое количество которых превосходит 400 (ключевое слово ***HAVING***…):

Замечание. Условие, что суммарное поставляемое количество должно быть больше 400 не может быть сформулировано в разделе WHERE, т.к. в этом разделе нельзя использовать агрегатные функции. Условия, использующие агрегатные функции должны быть размещены в специальном разделе HAVING:

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM
  HAVING SUM(PD.VOLUME) > 400;
```

В результате получим следующую таблицу:

DNUM	SM
1	1250
2	450

Замечание. В одном запросе могут встретиться как условия отбора строк в разделе WHERE, так и условия отбора групп в разделе HAVING. Условия отбора групп нельзя перенести из раздела HAVING в раздел WHERE. Аналогично и условия отбора строк нельзя перенести из раздела WHERE в раздел HAVING, за исключением условий, включающих поля из списка группировки GROUP BY.

Использование подзапросов

Очень удобным средством, позволяющим формулировать запросы более понятным образом, является возможность использования подзапросов, вложенных в основной запрос.

Пример 25. Получить список поставщиков, статус которых меньше максимального статуса в таблице поставщиков (сравнение с подзапросом):

```
SELECT *
  FROM P
  WHERE P.STATYS <
    (SELECT MAX(P.STATUS)
        FROM P);
```

Замечание. Т.к. поле P.STATUS сравнивается с результатом подзапроса, то подзапрос должен быть сформулирован так, чтобы возвращать таблицу, состоящую *ровно из одной строки и одной колонки*.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить максимальное значение статуса.
2. Просканировать таблицу поставщиков P, каждый раз сравнивая значение статуса поставщика с результатом подзапроса, и отобрать только те строки, в которых статус меньше максимального.

Пример 26. Использование предиката ***IN***. Получить список поставщиков, поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE P.PNUM IN
    (SELECT DISTINCT PD.PNUM
        FROM PD
        WHERE PD.DNUM = 2);
```

Замечание. В данном случае вложенный подзапрос может возвращать таблицу, содержащую несколько строк.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить список номеров поставщиков, поставляющих деталь номер 2.
2. Просканировать таблицу поставщиков P, каждый раз проверяя, содержится ли номер поставщика в результате подзапроса.

Пример 27. Использование предиката ***EXIST***. Получить список поставщиков, поставляющих деталь номер 2:

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

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Просканировать таблицу поставщиков P, *каждый раз выполняя подзапрос* с новым значением номера поставщика, взятым из таблицы P.
2. В результат запроса включить только те строки из таблицы поставщиков, для которых вложенный подзапрос вернул непустое множество строк.

Замечание. В отличие от двух предыдущих примеров, 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 EXIST, 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. Язык SQL является непроцедурным, а декларативным. Это значит, что пользователь, формулирующий запрос, просто описывает, *каким должен быть результат запроса*, а как этот результат будет получен - за это отвечает сама СУБД.

Пример 28. Использование предиката ***NOT EXIST***. Получить список поставщиков, не поставляющих деталь номер 2:

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

Замечание. Также 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.

Пример 29. Получить имена поставщиков, поставляющих все детали:

```
SELECT DISTINCT PNAME
  FROM P
  WHERE NOT EXIST
    (SELECT *
        FROM D
        WHERE NOT EXIST
          (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*.

Самый внутренний подзапрос параметризован двумя параметрами (D.DNUM, P.PNUM) и имеет следующий смысл: отобрать все строки, содержащие данные о поставках поставщика с номером PNUM детали с номером DNUM. Отрицание NOT EXIST говорит о том, что данный поставщик не поставляет данную деталь. Внешний к нему подзапрос, сам являющийся вложенным и параметризованным параметром P.PNUM, имеет смысл: отобрать список деталей, которые не поставляются поставщиком PNUM. Отрицание NOT EXIST говорит о том, что для поставщика с номером PNUM не должно быть деталей, которые не поставлялись бы этим поставщиком. Это в точности означает, что во внешнем запросе отбираются только поставщики, поставляющие все детали.

Использование объединения, пересечения и разности

Пример 30. Получить имена поставщиков, имеющих статус, больший 3 или поставляющих хотя бы одну деталь номер 2 (объединение двух подзапросов - ключевое слово ***UNION***):

```
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;
```

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.

Пример 31. Получить имена поставщиков, имеющих статус, больший 3 и одновременно поставляющих хотя бы одну деталь номер 2 (пересечение двух подзапросов - ключевое слово ***INTERSECT***):

```
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;
```

Пример 32. Получить имена поставщиков, имеющих статус, больший 3, за исключением тех, кто поставляет хотя бы одну деталь номер 2 (разность двух подзапросов - ключевое слово ***EXCEPT***):

```
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;
```

Синтаксис оператора выборки данных (SELECT)

BNF-нотация

Опишем синтаксис оператора выборки данных (оператора SELECT) более точно. При описании синтаксиса операторов обычно используются условные обозначения, известные как ***стандартные формы Бэкуса-Наура*** (***BNF***).

В BNF обозначениях используются следующие элементы:

- Символ "::=" означает равенство по определению. Слева от знака стоит определяемое понятие, справа - собственно определение понятия.
- Ключевые слова записываются прописными буквами. Они зарезервированы и составляют часть оператора.
- Метки-заполнители конкретных значений элементов и переменных записываются курсивом.
- 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.
- Многоточие, внутри которого находится запятая ".,.." указывает, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений. Запятую нельзя ставить после последнего элемента. Замечание: данное соглашение не входит в стандарт BNF, но позволяет более точно описать синтаксис операторов SQL.
- Круглые скобки являются элементом оператора.

Синтаксис оператора выборки

В довольно сильно упрощенном виде оператор выборки данных имеет следующий синтаксис (для некоторых элементов мы дадим не BNF-определения, а словесное описание):

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*}
]

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*]

Замечание. Select-выражение в разделе SELECT, используемое в качестве значения для отбираемого столбца, должно возвращать таблицу, состоящую из одной строки и одного столбца, т.е. скалярное выражение.

Замечание. Условное выражение в разделе WHERE должно вычисляться для каждой строки, являющейся кандидатом в результатирующее множество строк. В этом условном выражении можно использовать подзапросы. Синтаксис условных выражений, допустимых в разделе WHERE рассматривается ниже.

Замечание. Раздел HAVING содержит условное выражение, вычисляемое для каждой группы, определяемой списком группировки в разделе GROUP BY. Это условное выражение может содержать функции агрегирования, вычисляемые для каждой группы. Условное выражение, сформулированное в разделе WHERE, может быть перенесено в раздел HAVING. Перенос условий из раздела HAVING в раздел WHERE невозможен, если условное выражение содержит агрегатные функции. Перенос условий из раздела WHERE в раздел HAVING является плохим стилем программирования - эти разделы предназначены для различных по смыслу условий (условия для строк и условия для групп строк).

Замечание. Если в разделе SELECT присутствуют агрегатные функции, то они вычисляются по-разному в зависимости от наличия раздела GROUP BY. Если раздел GROUP BY отсутствует, то результат запроса возвращает не более одной строки. Агрегатные функции вычисляются по всем строкам, удовлетворяющим условному выражению в разделе WHERE. Если раздел GROUP BY присутствует, то агрегатные функции вычисляются по отдельности для каждой группы, определенной в разделе GROUP BY.

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**

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*

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 А [*Тип соединения*] **JOIN** *Таблица В* **USING** (Имя столбца.,..)

Соединение объединения ::=
Таблица А **UNION JOIN** *Таблица В*

Опишем используемые термины.

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 А и В, имеющим одинаковые имена. В результатирующую таблицу одинаковые столбцы вставляются только один раз.

JOIN … ON - Соединение посредством предиката соединяет строки таблиц А и В 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.

OUTER - Ключевое слово OUTER (внешний) не является обязательными, 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 А и В соединяются только те строки, для которых найдено совпадение.

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

RIGHT (OUTER) - Тип соединения "правое (внешнее)". Правое соединение таблиц А и В включает в себя все строки из правой таблицы В и те строки из левой таблицы А, для которых обнаружено совпадение. Для строк из таблицы В, для которых не найдено соответствия в таблице А, в столбцы, извлекаемые из таблицы А 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.

UNION JOIN - Соединение объединения является обратным по отношению к внутреннему соединению. Оно включает только те строки из таблиц А и В, 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.

Использование соединенных таблиц часто облегчает восприятие оператора SELECT, особенно, когда используется естественное соединение. Если не использовать соединенные таблицы, то при выборе данных из нескольких таблиц необходимо явно указывать условия соединения в разделе WHERE. Если при этом пользователь указывает сложные критерии отбора строк, то в разделе WHERE смешиваются семантически различные понятия - как условия связи таблиц, так и условия отбора строк (см. примеры 13, 14, 15 данной главы).

Синтаксис условных выражений раздела WHERE

Условное выражение, используемое в разделе WHERE оператора SELECT должно вычисляться для каждой строки-кандидата, отбираемой оператором SELECT. Условное выражение может возвращать одно из трех значений истинности: TRUE, FALSE или UNKNOUN. Строка-кандидат отбирается в результатирующее множество строк только в том случае, если для нее условное выражение вернуло значение TRUE.

Условные выражения имеют следующий синтаксис (в целях упрощения изложения приведены не все возможные предикаты):

Условное выражение ::=
[( ] [**NOT**]
{*Предикат сравнения*
| *Предикат between*
| *Предикат in*
| *Предикат like*
| *Предикат null*
| *Предикат количественного сравнения*
| *Предикат exist*
| *Предикат unique*
| *Предикат match*
| *Предикат overlaps*}
[{**AND** | **OR**} *Условное выражение*] [ )]
[**IS** [**NOT**] {**TRUE** | **FALSE** | **UNKNOWN**}]

Предикат сравнения ::=
Конструктор значений строки {= | < | > | <= | >= | <>} *Конструктор значений строки*

Пример 33. Сравнение поля таблицы и скалярного значения:

```
POSTAV.VOLUME > 100
```

Пример 34. Сравнение двух сконструированных строк:

```
(PD.PNUM, PD.DNUM) = (1, 25)
```

Этот пример эквивалентен условному выражению

```
PD.PNUM = 1 AND PD.DNUM = 25
```

Предикат between ::=
Конструктор значений строки [**NOT**] **BETWEEN**
Конструктор значений строки **AND** *Конструктор значений строки*

Пример 35. PD.VOLUME BETWEEN 10 AND 100

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

Пример 36.

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

Пример 37.

```
P.PNUM IN (1, 2, 3, 5)
```

Предикат like ::=
Выражение для вычисления значения строки-поиска [**NOT**] **LIKE**
Выражение для вычисления значения строки-шаблона [**ESCAPE** *Символ*]

Замечание. Предикат LIKE производит поиск строки-поиска в строке-шаблоне. В строке-шаблоне разрешается использовать два трафаретных символа:

- Символ подчеркивания "_" может использоваться вместо любого единичного символа в строке-поиска,
- Символ процента "%" может заменять набор любых символов в строке-поиска (число символов в наборе может быть от 0 и более).

Предикат null ::=
Конструктор значений строки **IS** [**NOT**] **NULL**

Замечание. Предикат NULL применяется специально для проверки, не равно ли проверяемое выражение null-значению.

Предикат количественного сравнения ::=
Конструктор значений строки {= | < | > | <= | >= | <>}
{**ANY** | **SOME** | **ALL**} (*Select-выражение*)

Замечание. Кванторы ANY и SOME являются синонимами и полностью взаимозаменяемы.

Замечание. Если указан один из кванторов ANY и SOME, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает *хотя бы с одним* значением, возвращаемом в подзапросе (select-выражении).

Замечание. Если указан квантор ALL, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает *с каждым* значением, возвращаемом в подзапросе (select-выражении).

Пример 38.

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

Предикат exist ::=
EXIST (*Select-выражение*)

Замечание. Предикат EXIST 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.

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.

Порядок выполнения оператора SELECT

Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

Стадия 1. Выполнение одиночного оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:

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 сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только 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.

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.

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, приведенном в разделе ORDER BY.

Как на самом деле выполняется оператор SELECT

Если внимательно рассмотреть приведенный выше концептуальный алгоритм вычисления результата оператора SELECT, то сразу понятно, что выполнять его непосредственно в таком виде чрезвычайно накладно. Даже на самом первом шаге, 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.

На самом деле 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:

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.

Шаг 2 (Преобразование в каноническую форму). Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании 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*.

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 из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, 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). На этом шаге 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.

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.

Оператор декартового произведения

Реляционная алгебра: $A \text{ TIMES } B$

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;
```

Оператор проекции

Реляционная алгебра: $A[X,Y,\ldots,Z]$

Оператор SQL:

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

Оператор выборки

Реляционная алгебра: $A \text{ WHERE } c$,

Оператор SQL:

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

Оператор объединения

Реляционная алгебра: $A \text{ UNION } B$

Оператор SQL:

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

Оператор вычитания

Реляционная алгебра: $A \text{ MINUS } B$

```
Оператор SQL:
SELECT *
    FROM A
    EXCEPT
    SELECT *
        FROM B
```

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. Тем не менее, для практических целей приведем их.

Оператор соединения

Реляционная алгебра: $(A \text{ TIMES } B) \text{ WHERE } c$

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;
```

Оператор пересечения

Реляционная алгебра: $A \,\mathrm{INTERSECT}\, B$

Оператор SQL:

```
SELECT *
    FROM A
    INTERSECT
    SELECT *
        FROM B;
```

Оператор деления

Реляционная алгебра: $A(X,Y)\,\mathrm{DEVIDBY}\, B(Y)$

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));
```

Замечание. Оператор SQL, реализующий деление отношений трудно запомнить, поэтому 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".

Преобразуем текст выражения:

"Отобрать номера поставщиков, которые поставляют *все* детали" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* непоставляемых деталей в таблице B" эквивалентно

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

"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 нужно учесть, что во внутреннем подзапросе таблица A должна быть переименована, для того чтобы отличать ее от экземпляра этой же таблицы, используемой во внешнем запросе.

Выводы

Фактически стандартным языком доступа к базам данных в настоящее время стал язык SQL (Structured Query Language).

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее.

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям:

- Операторы DDL (Data Definition Language) - операторы определения объектов базы данных.
- Операторы DML (Data Manipulation Language) - операторы манипулирования данными.
- Операторы защиты и управления данными, и др.

Одним из основных операторов DML является оператор SELECT, позволяющий извлекать данные из таблиц и получать ответы на различные запросы. Оператор SELECT содержит в себе все возможности реляционной алгебры. Это означает, что любой оператор реляционной алгебры может быть выражен при помощи подходящего оператора SELECT. Этим доказывается реляционная полнота языка SQL.

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 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.