ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ: СОЕДИНЕНИЕ, ОБЪЕДИНЕНИЕ, ВЫЧИТАНИЕ, ДЕКАРТОВОЕ ПРОИЗВЕДЕНИЕ. ПОДЗАПРОСЫ. ГРУППИРОВАНИЕ И АГРЕГАТНЫЕ ФУНКЦИИ. ПОСЛЕДОВАТЕЛЬНОСТИ. СИНОНИМЫ И ИХ СОЗДАНИЕ.

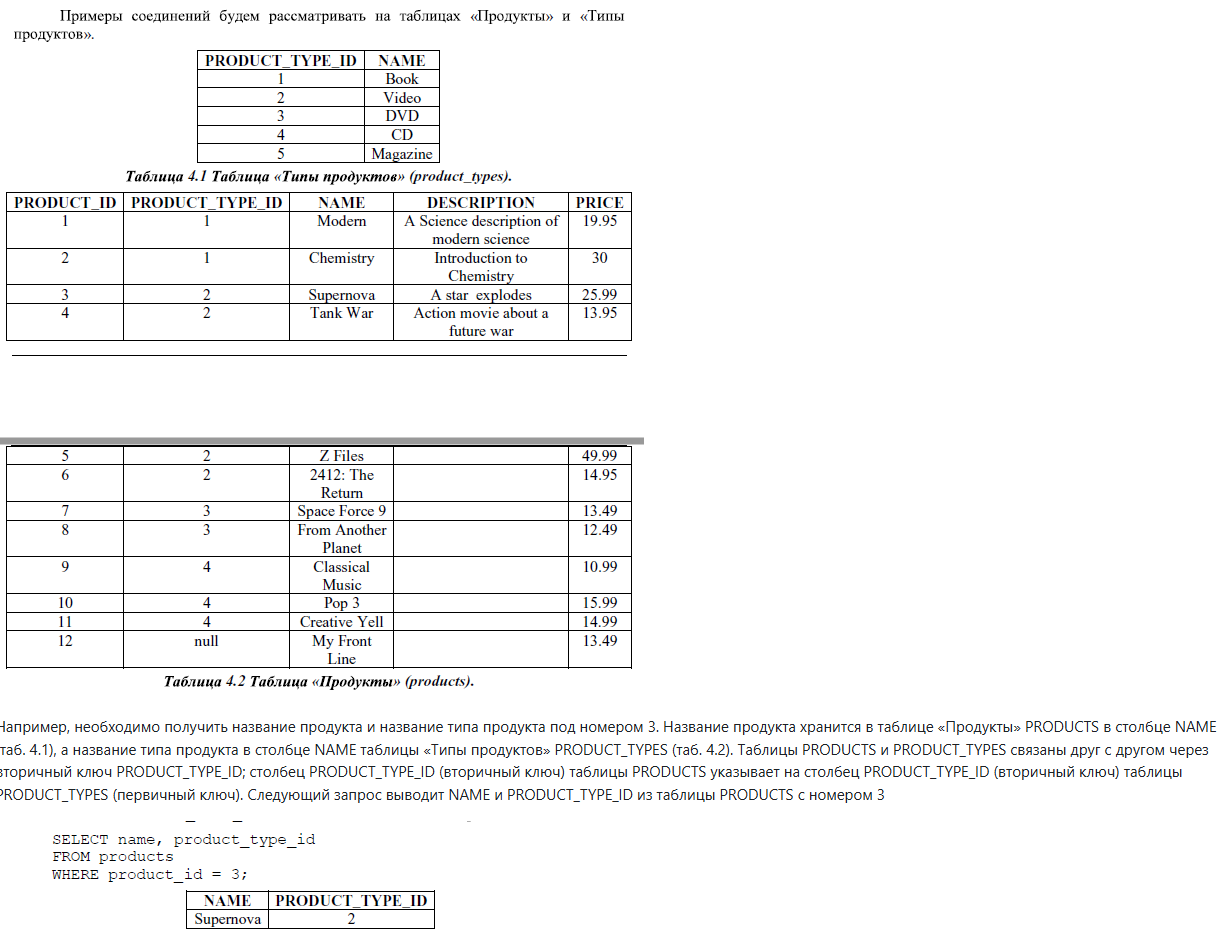
4.1 Теоретико-множественные операции.

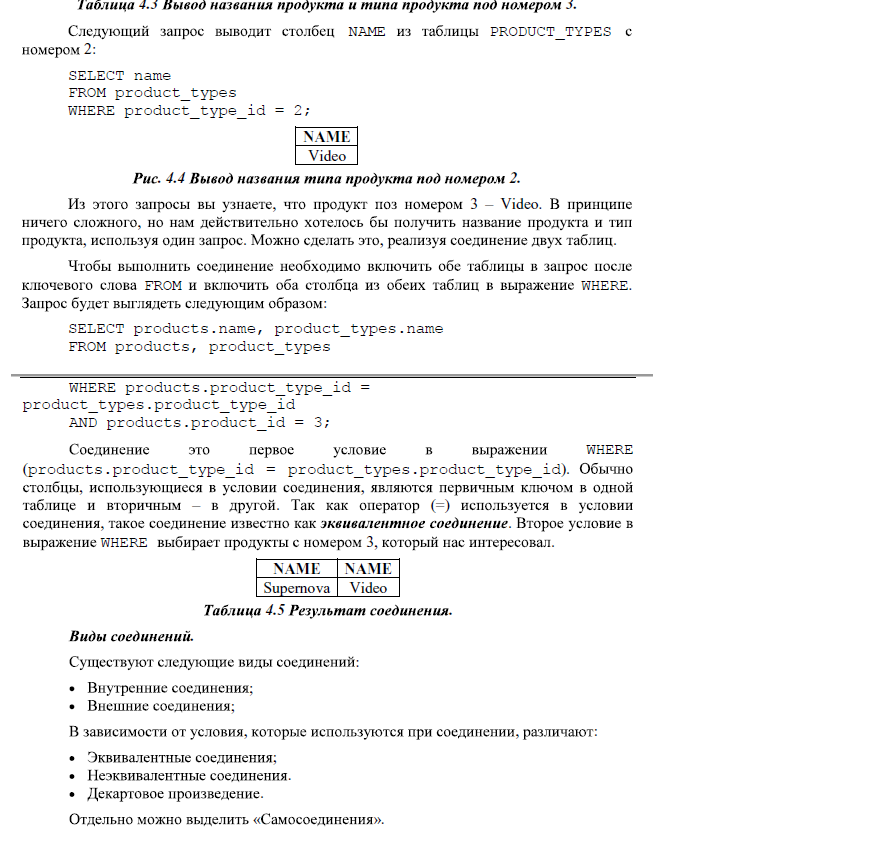
В ORACLE как и любой другой реляционной СУБД поддерживаются следующие виды теоретико-множественных операций:

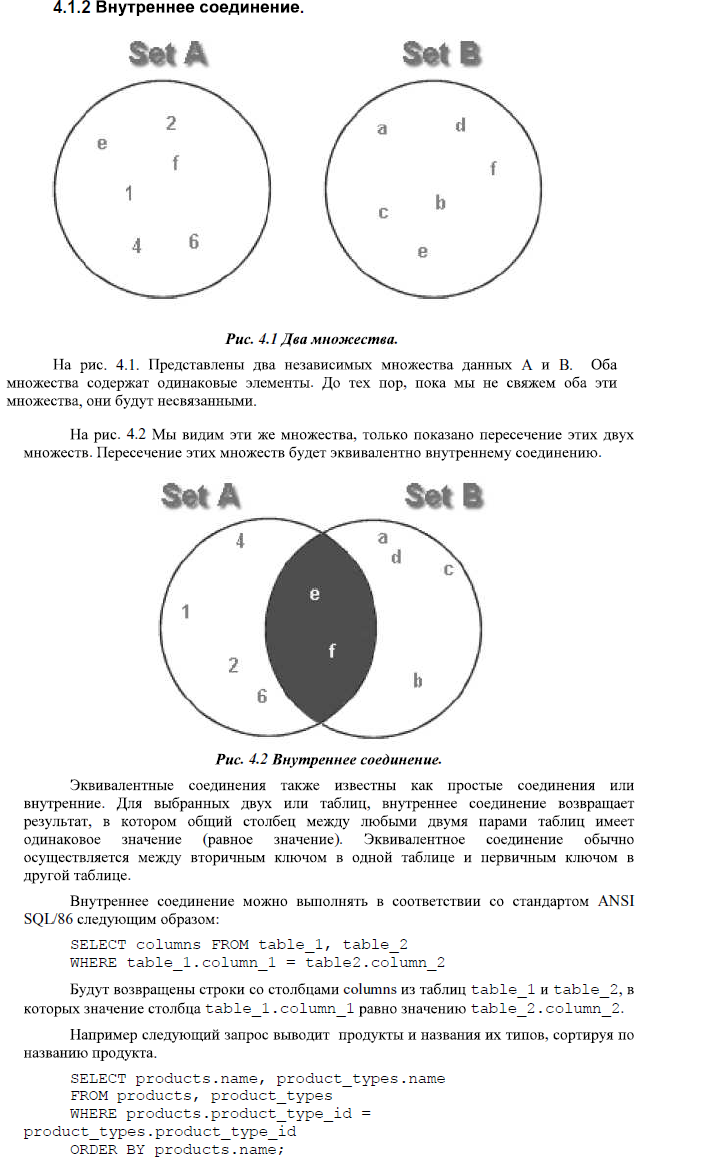
* соединение;
* декартовое произведение;
* объединение;
* вычитание;
* пересечение.

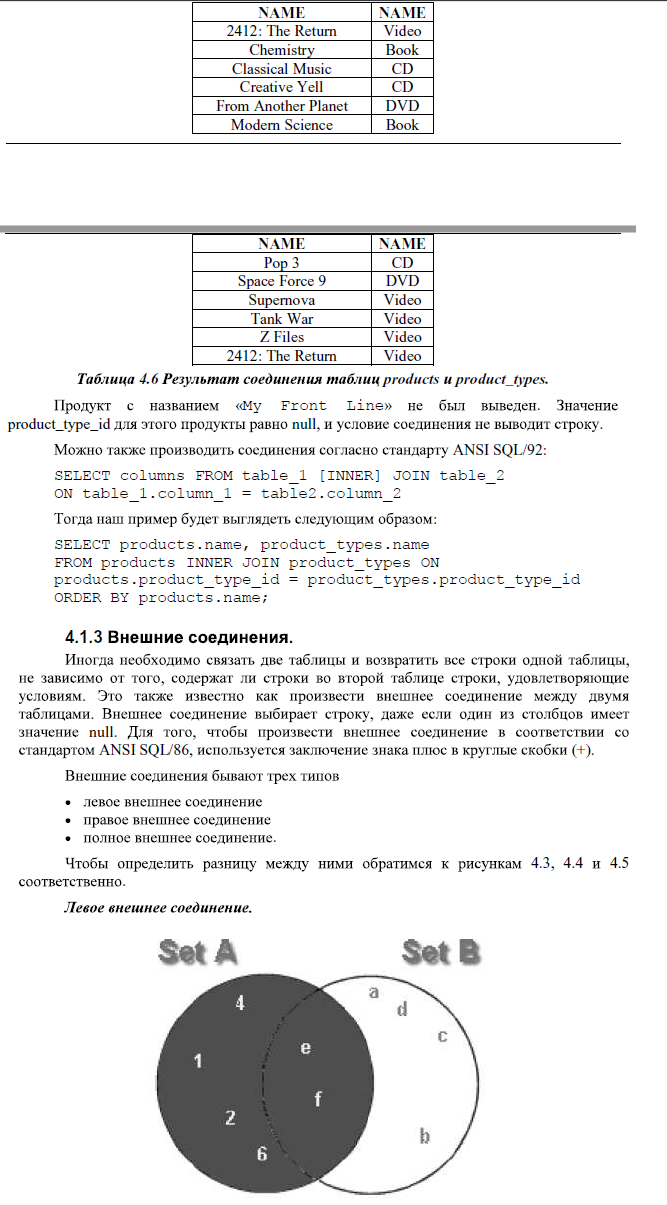
4.1.1 Соединения в ORACLE.

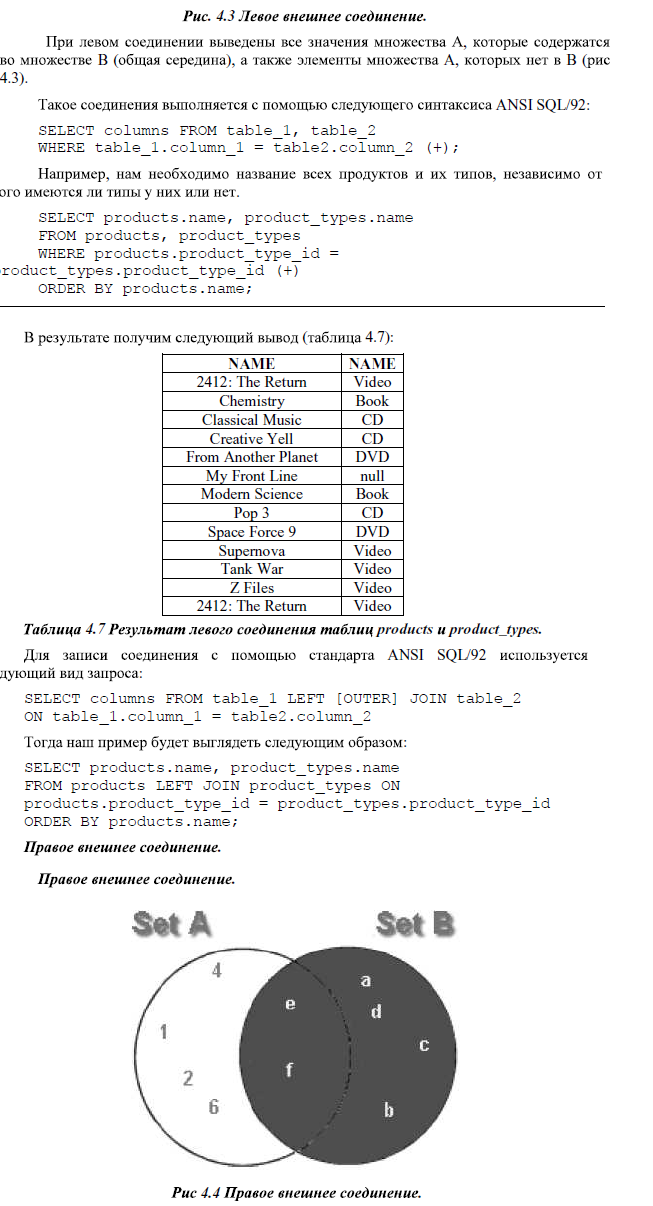
Схемы базы данных обычно содержат более чем одну таблицу, которые хранят различные данные. Например, схема «Store» имеет таблицы, которые содержат информацию о клиентах, продуктах, работниках и др. Все запросы, показанные в предыдущих лекциях, запрашивали и выводили информацию только из одной таблицы. В реальности же необходимо запрашивать и получать информацию из нескольких таблиц.

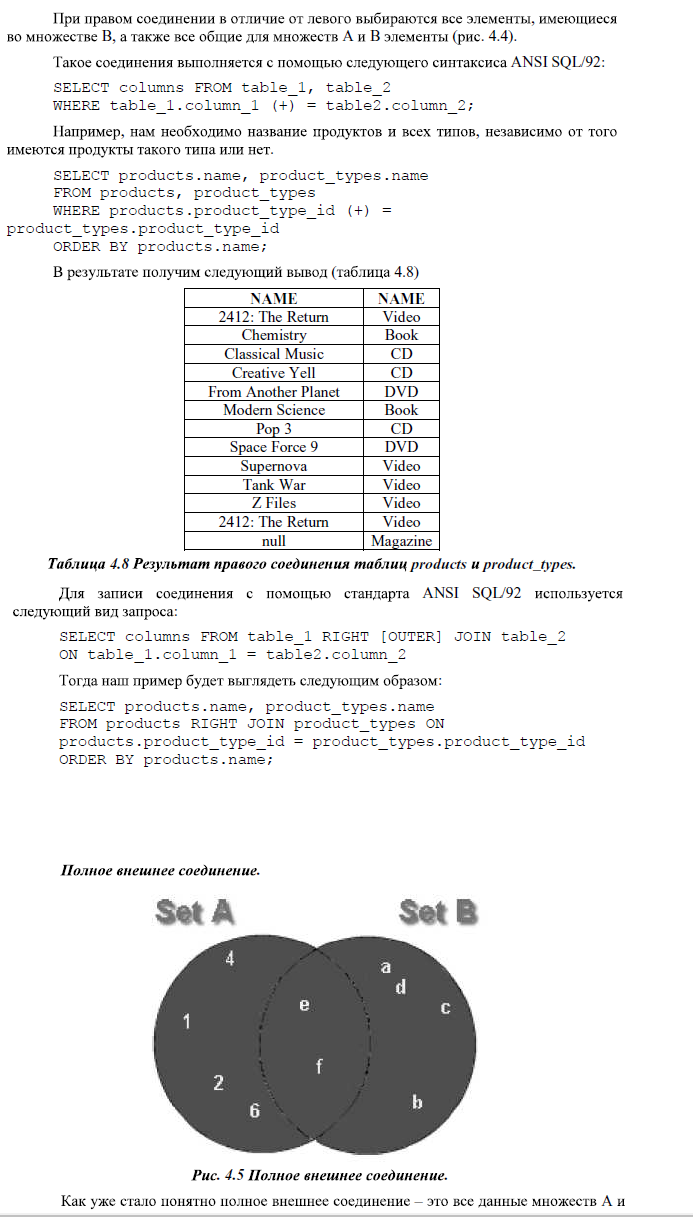


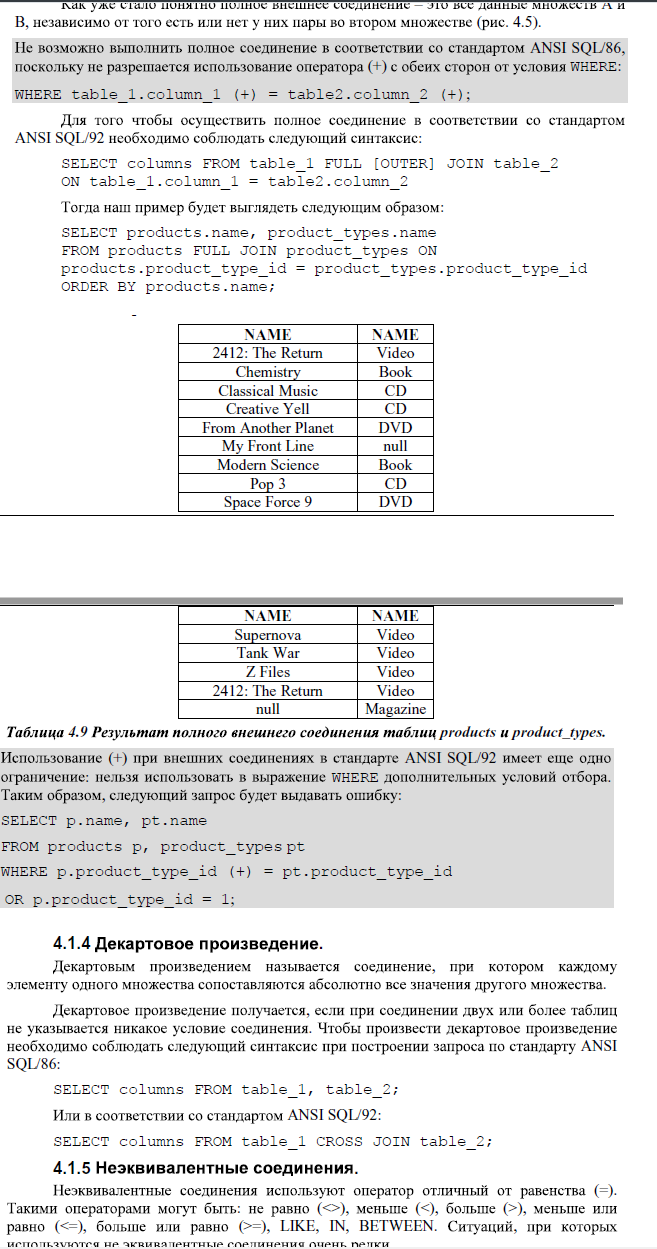


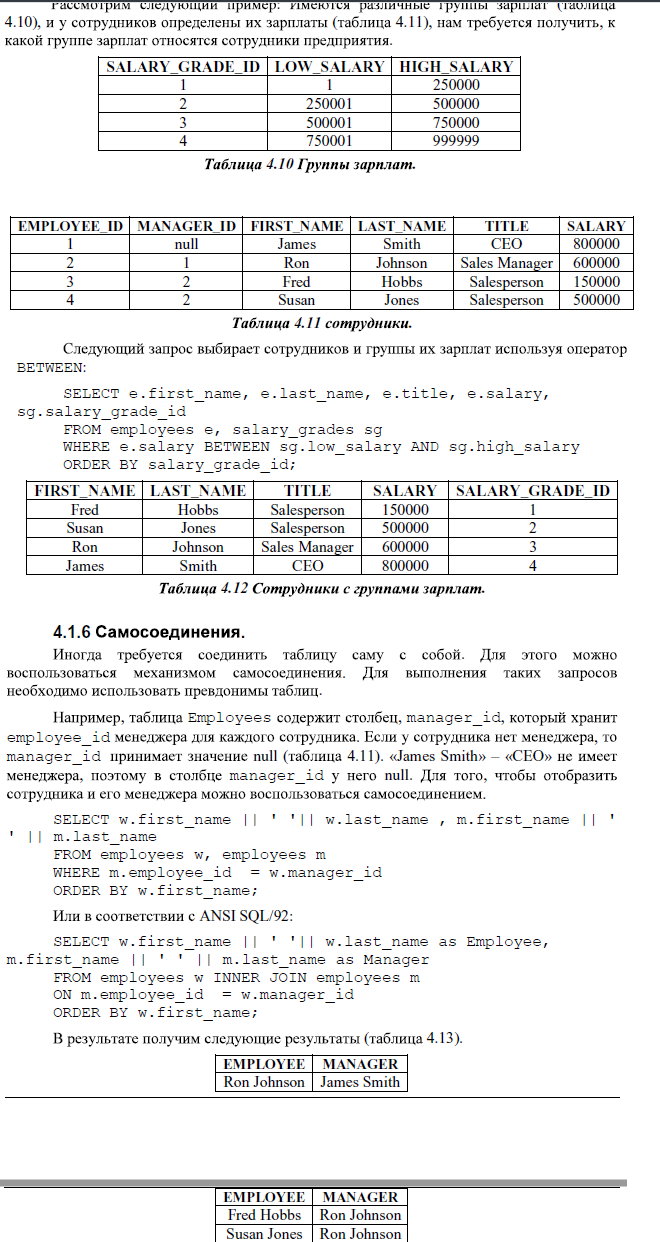


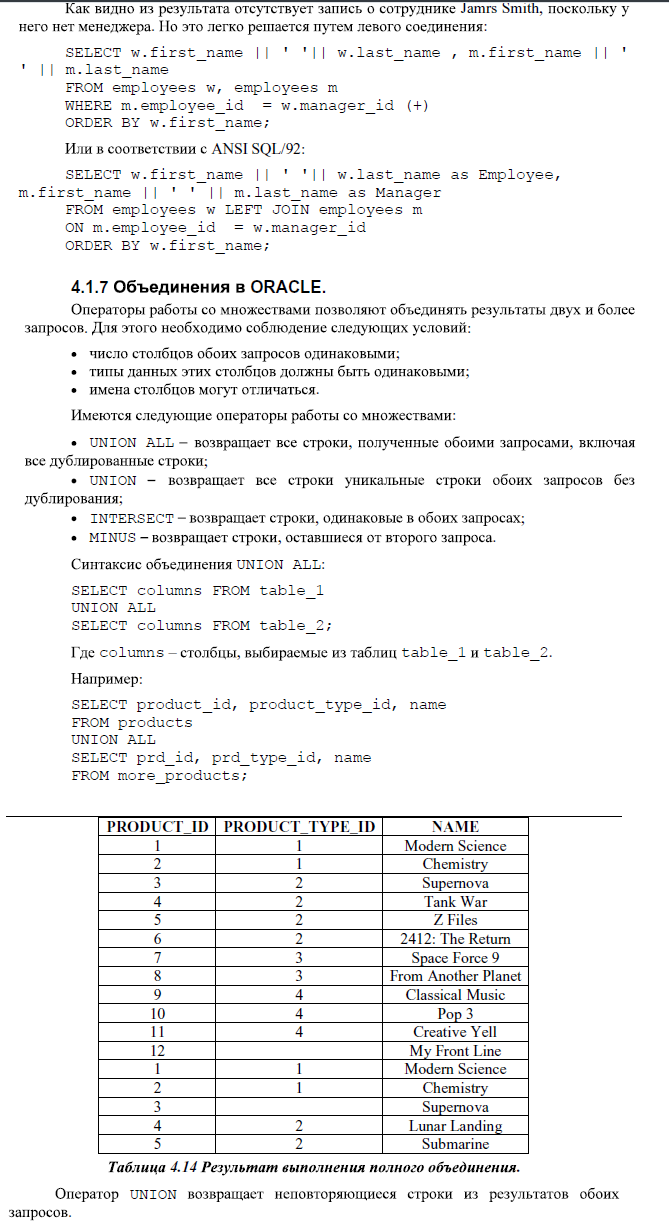


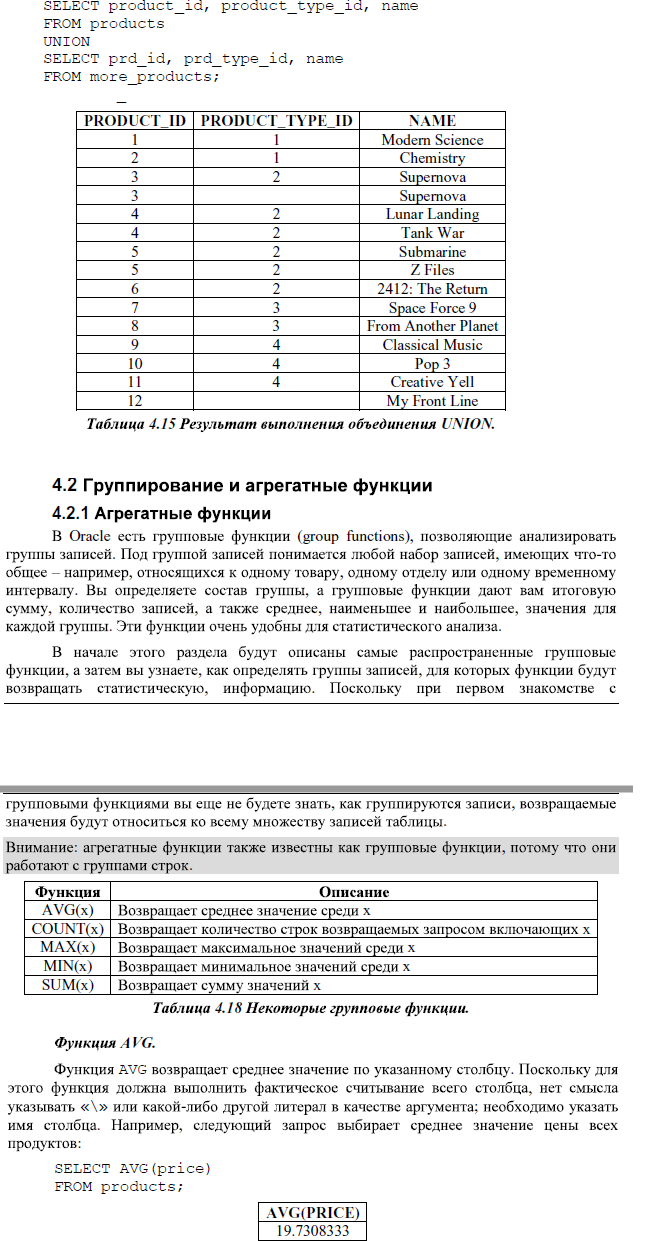


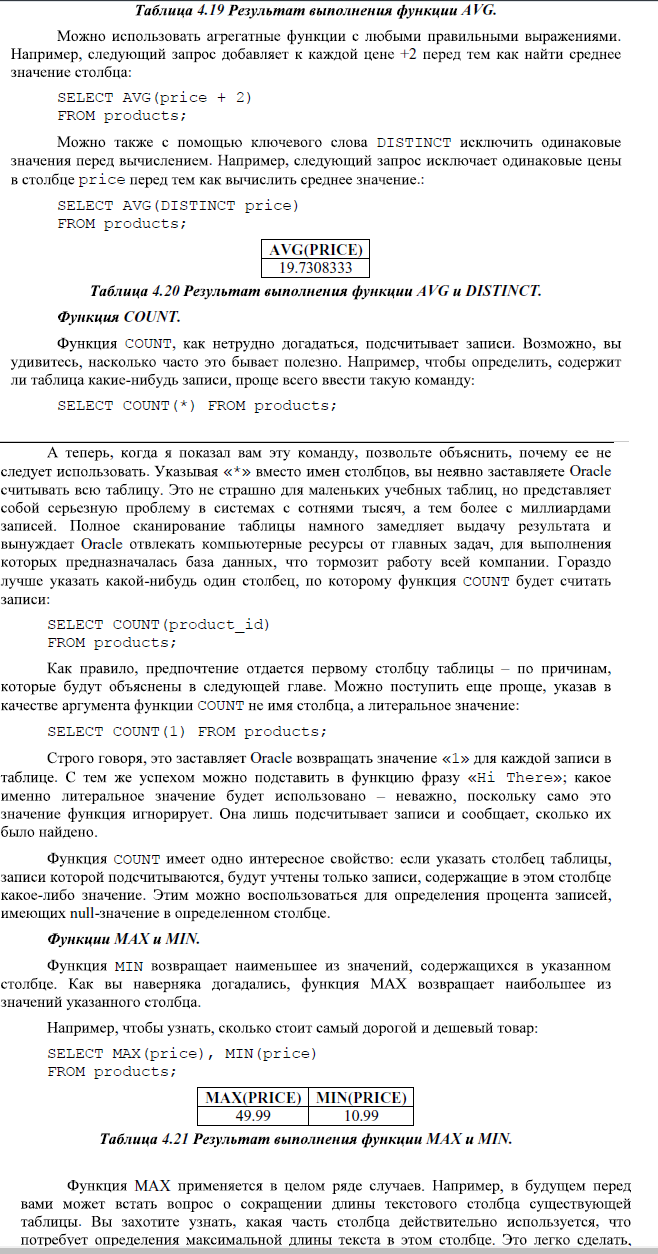


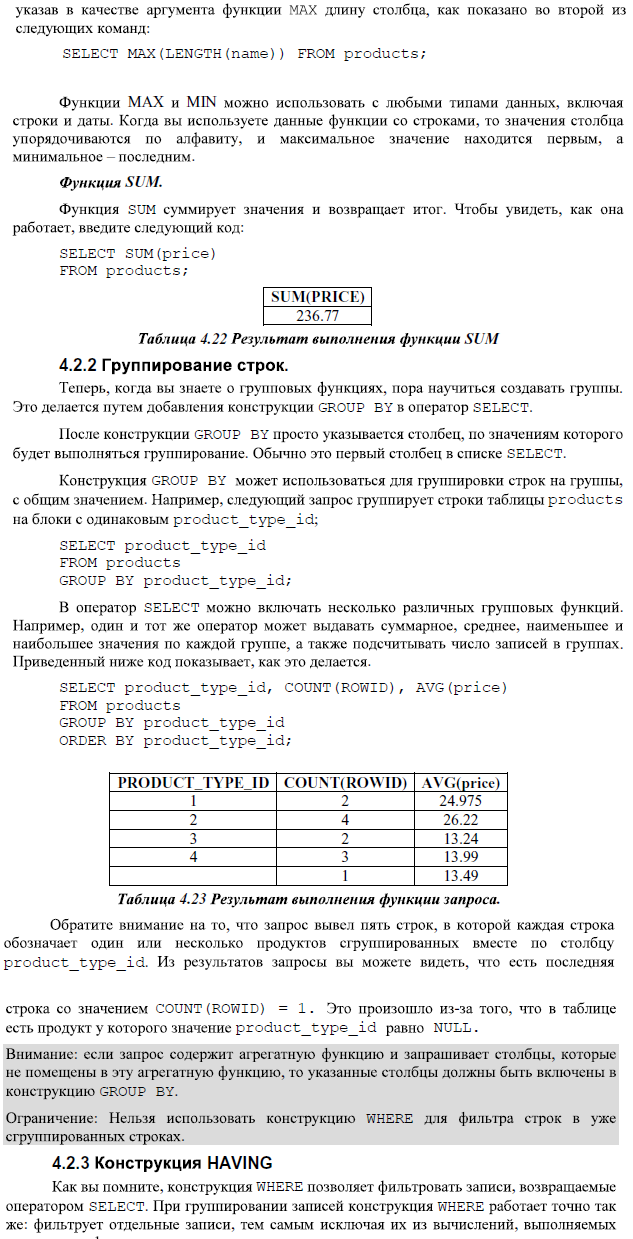


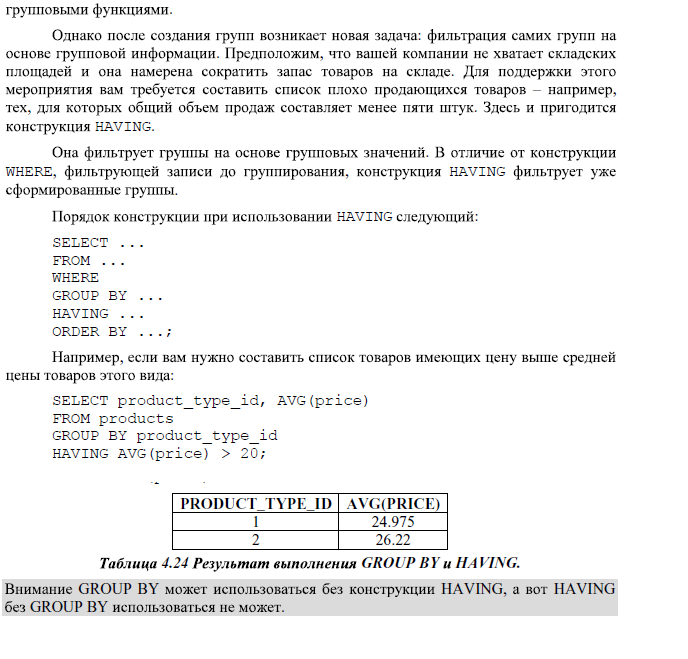












4.3 Подзапросы.

Подзапросы помещают один запрос в другой. Второй запрос помещается внутрь WHERE для расширения запросов SELECT. Один или несколько значений, возвращаемых вложенным запросом, используется в основном запросе в качестве фильтра.

Типы операторов, используемых в выражении WHERE, зависит от того одно или несколько значений возвращается подзапросом. Если возвращается только одно значение вложенным запросом, используется оператор сравнения (=, !=, <, >, <=, >=) и др. Если же возвращается более одного значения, то используется такие операторы как IN, ANY, ALL,

NOT.

Существуют 2 основных типа подзапросов:

* Нормальные подзапросы. Замкнутые запросы, предполагающие, что прямой связи между вызываемым подзапросом и запросом нет.
* Соотнесенные (корреляционные) подзапросы. Слово корреляция используется, чтобы описать связь между запросом и вызываемым подзапросом. Необходимо помнить об основном правиле, что корреляционный столбец из внешнего запроса должен быть внесен в подзапрос, но не наоборот. Таким образом, соотнесенный (корреляционный) подзапрос всегда зависит от основного запроса.

Подзапросы также можно разделить на вложенные и встраиваемые представления .

* Вложенные подзапросы. Подзапросы могут вызывать другие подзапросы и так без конца. Другими словами, подзапросы могут вкладываться в другие подзапросы.
* Встраиваемые представления (inline views). Встраиваемые представления это подзапросы, путем встраивания в предложение FROM запроса SELECT, который, кстати, тоже является подзапросом. Значения могут быть переданы из встраиваемого представления во внешний запрос или подзапрос.

Подзапросы по количеству возвращаемы строк могут разделяться на:

* Однострочные подзапросы, возвращающие одну строку во внешний запрос или не возвращающие ни одной строки. Есть один случай, когда однострочный подзапрос возвращает один столбец, такой подзапрос называют скалярным.
  + Многострочные подзапросы возвращают несколько строк во внешний запрос.

Подзапросы можно вставлять почти везде в любом месте SQL, в любой SQL оператор, который поддерживает выражения. Далее представлены наиболее распространенные места SQL, в которых может быть помещен подзапрос:

* После ключевого слова SELECT в местах определения столбцов запросы.
* После ключевого слова WHERE как условие запроса.
* После ключевого слова FROM (вложенное представление).
* В месте определения новых значений после ключевого слова VALUES оператора

INSERT.

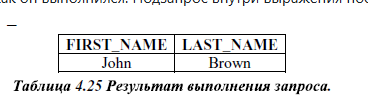
* В операторе UPDATE следующим образом SET clause = (подзапрос).
* В качестве формального параметра процедуры или функции.

4.3.1 Однострочные подзапросы

Вы можете вставить подзапрос в выражение после ключевого слова WHERE внешнего запроса. Пример

SELECT first\_name, last\_name FROM customers WHERE customer\_id = (SELECT customer\_id FROM customers WHERE last\_name = 'Brown');

Этот пример получает данные first\_name и last\_name строки из таблицы customers, у которой значение last\_name равно Brown. Давайте разобьем этот запрос и проанализируем, как он выполнился. Подзапрос внутри выражения после ключевого



слова WHERE:

SELECT customer\_id

FROM customers

WHERE last\_name = 'Brown';

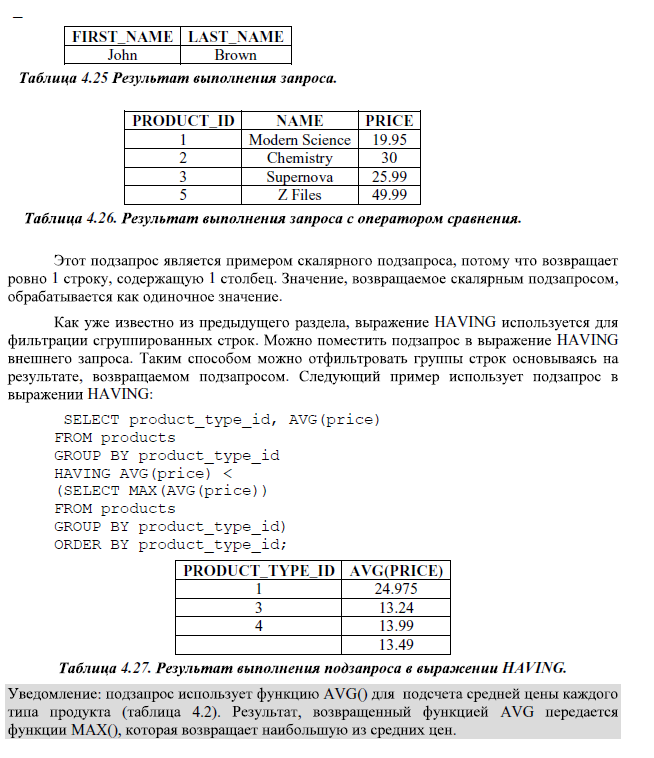
Этот подзапрос выполняется сначала (и только один раз) и возвращает customer\_id для строки, у которой значение last\_name равно Brown. Значение customerid для этой строки равно 1, которое подставляется в выражение поле ключевого слова WHERE внешнего запроса. Таким образом, внешний запрос возвращает такой же результат, что и следующий запрос:

SELECT first\_name, last\_name

FROM customers

WHERE customer\_id = 1;

Предыдущий запрос использовал в выражении после ключевого слова WHERE оператор эквивалентности (=). Вы также можете использовать другие операторы сравнения, такие как <>, <, >, <=, >=, с однострочными подзапросами. Следующий пример использует > в выражении WHERE внешнего запроса; подзапрос использует функцию AVG() чтобы получить среднюю цену всех продуктов, которая будет помещена в выражение после ключевого слова WHERE во внешнем запросе. Весь запрос выбирает номер product\_id, название name и цену price продуктов, имеющих цену больше средней цены всех продуктов:



4.3.2 Многострочные подзапросы

Многострочные запросы возвращают несколько строк. Операторы IN EXISTS, а также группа операторов сопоставления ANY, ALL, SOME позволяют работать с результатами многострочных подзапросов.

Возможности этих операторов приведены ниже:

* Множественный подзапрос может возвратить множество значений для сравнения оператора IN. Как вы помните оператор IN проверяет есть ли такое значение во множестве. Следующий пример выводит product\_id и name из таблицы products, если product\_id является одним из продуктов содержащим в названии букву «e»

SELECT product\_id, name

FROM products

WHERE product\_id IN

(SELECT product\_id

FROM products

WHERE name LIKE '%e%');

* Сопоставление EXIST обычно использует индексы для нахождения пары, для значения в подзапросе со значением из внешнего запроса. Не обращая внимания на корреляционный индекс столбца между внешним запросом и подзапросом, EXIST
* останавливает выполнения подзапроса, когда хотя бы одно соответствующее значение найдено. IN построит все значения множества, возвращаемого подзапросом, перед тем, как передать результаты во внешний запрос. Предпочтительное использование EXIST вместо IN часто выражается в лучшей производительности. EXIST может не работать производительнее, если множеством, возвращаемым подзапросом, является ограниченное множество литеральных значений или малое количество строк.

• ANY, ALL, и SOME подразумевают любое, все или некоторое количество значений соответственно. Из-за того, что подзапрос сопоставляется со множеством значений, подзапрос может возвращать как в реальности нуль, одну или множество строк.

Следующий запрос использует оператор ANY для того, чтобы получить сотрудников, у которых зарплата ниже самого низкого значения зарплаты в таблице salary\_grades.

* Внимание: Очень важно, чтобы многострочный подзапрос мог возвращать нуль строк, поскольку вхождение (IN), существование (EXIST) и соотнесение (ANY, ALL, SOME) возвращает множество значений. Пустое множество также может быть множеством значений. К тому же пустое множество является правильным множеством.

Подзапросы можно помещать после ключевого слова FROM внешнего запроса. Этот тип подзапросов также известен как, встраиваемым представлением, потому что подзапрос представляет данные в качестве таблицы.

Например, получить название продуктов из таблицы продуктов, чей product\_id меньше 3:

SELECT product\_id

FROM

(SELECT \* FROM products

WHERE product\_id < 3);

Следующий пример более является более наглядным: возвратить product\_id и price из таблицы products во внешнем запросе, и подзапросом получить количество раз данный продукт был заказан:

SELECT prds.product\_id, price, purchases\_data.product\_count FROM products prds,

(SELECT product\_id, COUNT(product\_id) product\_count FROM purchases

GROUP BY product\_id) purchases\_data

4.3.3  Ошибки, возникающие при использовании подзапросов

* 1. Однострочный подзапрос возвращает больше одной строки

SELECT product\_id, name FROM products

WHERE product\_id = (SELECT product\_id FROM products WHERE name LIKE '%e%');

* 1. Подзапрос содержит конструкцию ORDER BY.

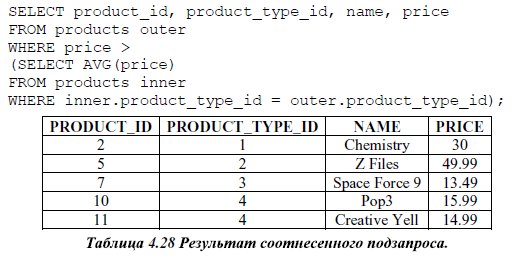
SELECT product\_id, name, price FROM products

WHERE price > (SELECT AVG(price) FROM products ORDER BY product\_id DESC);

4.3.4  Соотнесенные подзапросы.

Соотнесенный подзапрос ссылается на один или несколько столбцов во внешнем запросе. Такие подзапросы еще называют корреляционными. Чаще всего соотнесенные подзапросы используются, когда необходимо ответить на вопрос, который зависит от содержащегося значения каждой строки внешнего запроса.

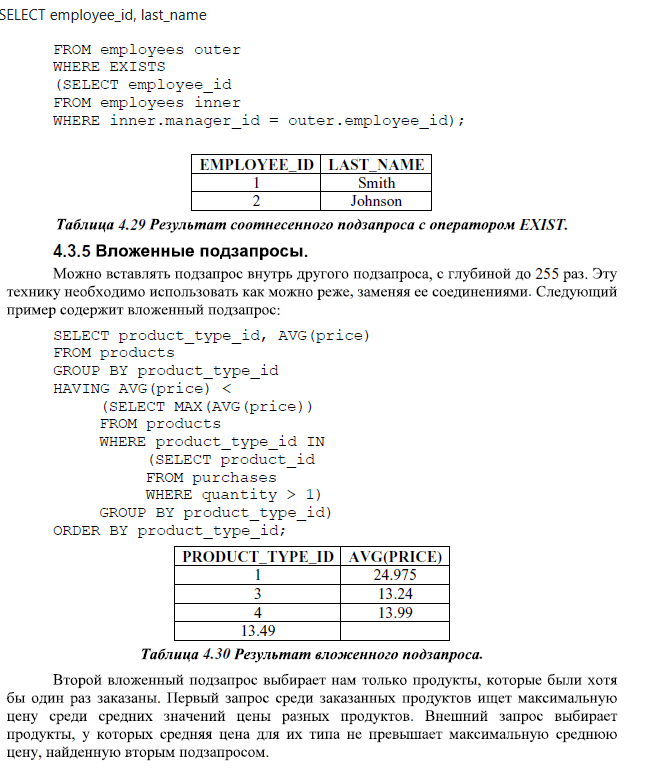
Следующий соотнесенный подзапрос получает информацию о продуктах которые имеют цену больше средней цены для данного типа продукта:



Обратите внимание, что в примере использованы псевдонимы outer для внешнего запроса и inner для подзапроса. В соотнесенном подзапросе каждая строка внешнего запроса имеет связь со строкой подзапроса. Подзапрос считывает каждую строку внешнего запроса и применяет ее значения для своего вычисления, до тех пор, пока не будет считана последняя строка. После чего формируется результат.

Можно использовать оператор EXIST для проверки существования строк, возвращаемых подзапросом.

Следующий пример показывает использование EXIST для получения списка сотрудников, которые управляют другими сотрудниками:



4.3.6 Использование подзапросов с операторами UPDATE и

DELETE

Можно также использовать подзапросы внутри операторов UPDATE и DELETE.. В запросе с оператором UPDATE подзапросом можно обновить значение столбца в том случае, если подзапрос возвращает одно значение, т.е. является однострочным. Например, изменить зарплату сотрудника под номером 4 в таблице employee на среднее значение столбца high\_halary таблицы salary\_grades:

UPDATE employees

SET salary =

(SELECT AVG(high\_salary)

FROM salary\_grades) WHERE employee\_id = 4;

1 row updated.

Можно использовать строки, возвращаемые подзапросом в конструкции WHERE оператора DELETE. Например, можно удалить сотрудников, чья заработная плата больше среднего значения high\_salary таблицы salary\_grades:

DELETE FROM employees WHERE salary > (SELECT AVG(high\_salary) FROM salary\_grades); 1 row deleted.

Внимание: Если вы используете операторы UPDATE и DELETE не забывайте выполнять ROLLBACK или COMMIT чтобы восстановить или зафиксировать транзакцию.

4.4 Последовательности синонимы и их создание. 4.4.1 Последовательности

Базы данных предназначены для поддержания порядка в массивах информации. Одним из способов упорядочения записей является присваивание им последовательных номеров. Oracle позволяет создавать счетчики, называемые последовательностями (sequences), которые увеличиваются каждый раз, когда к ним происходит обращение. Ссылаясь на последовательность при вставке записей, можно гарантировать, что каждой записи будет присвоен новый уникальный номер.

Для создания последовательности используется следующий синтаксис:

CREATE SEQUENCE <имя\_последовательности>;

Эта простая команда создает последовательность, которая начинается с 1 увеличивается на 1 при каждом обращении. Ничего другого от последовательностей часто и не требуется. Тем не менее, при определении последовательности можно использовать много дополнительных параметров. Взгляните на приведенный ниже синтаксис, где перечислены наиболее полезные из них:

CREATE SEQUENCE <имя\_последовательности> [INCREMENT BY <значение\_инкремента>] [START WITH <начальное\_значение>] [MAXVALUE <наибольшее\_значение>] [MINVALUE <наименьшее\_значение>]

[CYCLE]

Параметр INCREMENT BY позволяет создавать последовательности с инкрементом, отличным от 1. Значение этого параметра может содержать до 28 цифр (хотя трудно представить ситуации, где может найти применение такой инкремент!). Если указать здесь отрицательное число, то значение последовательности будет уменьшаться при каждом обращении.

создать   последовательность, начальное значение которой отлично от 1 . Это может пригодиться при создании последовательности для таблицы, уже содержащей записи, чтобы начать последовательность с числа, следующего за наибольшим существующим идентификатором записи.

Параметры MAXVALUE и MINVALUE позволяют ограничить интервал чисел, генерируемых последовательностью. Если использовать их в сочетании с параметром CYCLE, заданное множество значений будет циклически повторяться. Наиболее распространены последовательности с инкрементом 1 и без ограничений на генерируемые значения. Создайте такую последовательность с помощью приведенной ниже команды и переходите к следующему разделу, где будет рассмотрено ее использование.

CREATE SEQUENCE test\_seq;

Чтобы получать значения последовательности, на нее необходимо ссылаться как на таблицу. Последовательности содержат два "псевдостолбца" с именами CURRVAL и NEXTVAL, которые возвращают текущее и следующее значения последовательности соответственно. Выборка из столбца NEXTVAL вызывает автоматический инкремент последовательности.

Чтобы увидеть, как это происходит, введите следующие команды и сравните:

SELECT test\_seq.nextval FROM DUAL; SELECT test\_seq.nextval FROM DUAL; SELECT test\_seq.nextval FROM DUAL;

Теперь вам нужно научиться заполнять столбцы таблицы из последовательности. Это делается путем включения ссылки на последовательность в оператор INSERT, как в приведенных ниже командах74

CREATE TABLE test (

record\_id NUMBER (18, 0), record\_text VARCHAR2 (10)

);

INSERT INTO test VALUES  (test\_seq.nextval, 'Record A');

INSERT INTO test VALUES  (test\_seq.nextval, 'Record B');  
SELECT \* FROM test;

Примечание: Хотя последовательности обычно создаются для одной таблицы, в Oracle нет ограничения, которое бы этого требовало. Последовательность является независимым объектом. Она может использоваться в одной таблице, во многих таблицах или ни в одной из таблиц.

В методе, который вы только что использовали, демонстрировалось обращение к последовательности при помощи явной ссылки в операторе INSERT. Можно также организовать автоматическое обращение к последовательности, чтобы не ссылаться на нее в операторе INSERT.

Созданную последовательность можно модифицировать различными способами. В частности, можно изменить значение инкремента, скорректировать или удалить минимальное и максимальное значения, разрешить или запретить циклический повтор по достижении граничного значения.

Синтаксис,    позволяющий    выполнять    эти    изменения    в существующей последовательности, очень похож на синтаксис, используемый для ее создания. Он выглядит следующим образом:

ALTER SEQUENCE <имя последовательности^ [INCREMENT BY <значение\_инкремента>] [MAXVALUE <наибольшее\_значение>   | NOMAXVALUE] [MINVALUE <наименыиее\_значение>   | NOMINVALUE]

[CYCLE | NOCYCLE] 4.4.2 Синонимы

Синоним (synonym) позволяет ссылаться на объект Oracle по имени, которое отличается от его настоящего имени. Синонимы можно определять для таблиц, представлений, последовательностей, а также других объектов - функций, процедур и пакетов. В этом разделе речь пойдет о синонимах таблиц, но все сказанное ниже применимо и к синонимам других объектов.

Зачем нужно создавать синоним для какого-либо объекта? Главным образом для удобства: если вы часто ссылаетесь на таблицу с длинным именем, то по достоинству оцените возможность использования короткого имени без переименования таблицы и изменения кода, который на нее ссылается.

Удобство синонимов проявляется и в том, что они могут облегчить доступ к вашим данным для других людей. Таблицы организуются по идентификатору пользователя Oracle, который их создает, поэтому если другой пользователь захочет обращаться к таблице, созданной вами, то в общем случае ему придется помещать перед именем таблицы ваше имя пользователя, как показано ниже:

SELECT \* FRC^ <ваше\_имя\_пользователя>.<имя\_вашей\_таблицы>;

Это может оказаться утомительным занятием, а таблицу кому-то другому, то вдобавок потребуется менять весь код, который на нее ссылается. Синонимы позволяют сделать таблицу "видимой" для всех, даже если не указано имя ее владельца. Благодаря этому можно писать SQL-операторы, которые будут продолжать работать даже при передаче таблицы другому пользователю.

Создание синонима

Команда создания синонима имеет следующий синтаксис:

CREATE   [PUBLIC]   SYNONYM <имя\_синонима> FOR <имя\_объекта>

Если вам просто нужно сделать таблицу доступной другим пользователям, создайте синоним с тем же именем, что и у таблицы. Вот пример команды такого типа:

CREATE PUBLIC SYNONYM product FOR product; Модификация существующего синонима.

Ввиду чрезвычайной простоты синонимов Oracle не предоставляет никаких средств для их изменения. При необходимости просто удалите старый синоним и создайте новый. Команда удаления синонима имеет следующий синтаксис:

DROP   [PUBLIC]   SYNONYM <имя\_синонима>;

Чтобы удалить первый из созданных выше синонимов (PROD), введите следующую команду:

DROP SYNONYM prod;

Для удаления общего синонима введите такую команду: DROP PUBLIC SYNONYM product;