Методические указания к заданию №1.

Задание выполняется в 1 семестре состоит из 3-х частей:

1. Создание реляционной модели базы данных;

2. Создание физической модели базы данных в СУБД Oracle;

3. Написание SQL запросов к базе данных.

(Вариант задания приведен в конце методических указаний.)

1 Создание реляционной модели базы данных для заданной предметной области

Краткие теоретические сведения:

Модель «Сущность-связь» (ER) была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных.

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

Ключ сущности - это неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из ключа

нарушается его уникальность.

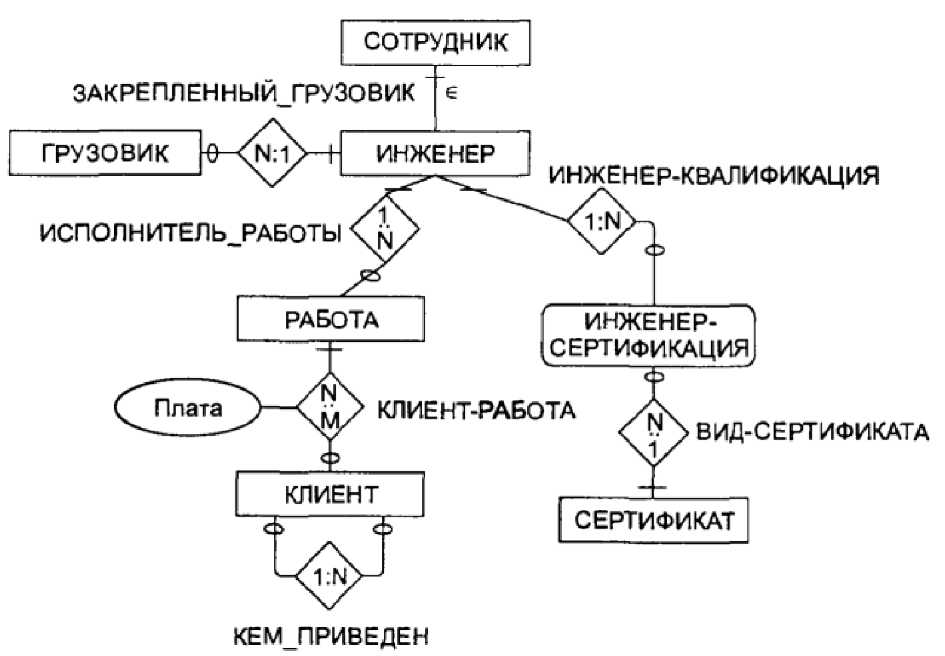
Связь - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Каждая связь может иметь один из следующих типов связи:

«один-к-одному» означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две.

«один-ко-многим» или «многие-к-одному» означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи. Левая сущность (со стороны «один») называется родительской, правая (со стороны «много») -дочерней.

«много-ко-многим» означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.



КЕМ\_ПРИВЕДЕН

Рис 1.1 Пример диаграммы сущность-связь.

Теоретической основой реляционной модели стала теория отношений, основу которой заложили два логика — американец Чарльз Содерс Пирс (1839-1914) и немец Эрнст Шредер (1841-1902). Основоположником реляционных баз данных считается Эдгар Кодд, который в 1970 году опубликовал в одном журнале статью «Реляционная модель данных для больших банков данных совместного использования». В статье Эдгар рассказывал про новую систему управления данными, основанную на математических принципах теории множеств и теории предикатов. Такая модель поддерживала точность и непротиворечивость данных, а также удобное извлечение и модификацию данных, со структурой, не зависящей от приложений и платформ.

Доменом называется множество атомарных значений одного и того же типа. Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа.

Вхождение домена в отношение принято называть атрибутом.

Схема отношения - это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Попросту говоря, кортеж - это набор именованных значений заданного типа.

Ключ (key) - это группа из одного или более атрибутов, которая уникальным образом идентифицирует строку.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения.

Обычным житейским представлением отношения является двумерная таблица. Каждая строка в таблице содержит данные, относящиеся к некоторой вещи или какой-то ее части. Каждый столбец таблицы описывает какой-либо атрибут этой вещи. Иногда строки называются кортежами (tuples), а столбцы — атрибутами (attributes).

2 Создание реляционной базы данных с использованием языка определения данных

Краткие теоретические сведения:

Data Definition Language (DDL) (язык описания данных) — это семейство компьютерных языков, используемых в компьютерных программах для описания структуры баз данных. В базах данных DDL является подмножеством SQL, используемым для определения и модификации различных структур данных.

К данной группе относятся команды, предназначенные для создания, изменения и удаления различных объектов базы данных. Команды CREATE (создание), ALTER (модификация) и DROP (удаление) работают с большинством типов объектов баз данных (таблиц, представлений, процедур, триггеров, табличных областей, пользователей и др.).

После создания общей структуры базы данных можно приступить к созданию таблиц, представляющих отношения, входящие в состав проекта базы данных. Для этой цели используется оператор CREATE TABLE, имеющий следующий общий формат:

CREATE TABLE [schema.] table

( column datatype [DEFAULT expr] [column\_constraint]

[, { column datatype [DEFAULT expr] [column\_constraint] ...

| CONSTRAINT constraint }]...);

Описание column\_constraint:

[NOT NULL]

[PRIMARY KEY]

[FOREIGN KEY REFERENCES table (columns) [ON {UPDATE | DELETE} { CASCADE | SET DEFAULT | SET NULL |NO ACTION }]]

[UNIQUE]

[CHECK (condition)]

Эта версия оператора CREATE TABLE включает средства определения

ограничений ссылочной целостности и других ограничений. В результате

выполнения этого оператора будет создана таблица, имя которой определяется

параметром table, состоящая из одного или нескольких столбцов column типа

datatype. Набор доступных типов данных описан ниже.

Для задания значения, применяемого по умолчанию при вставке данных в

конкретный столбец, предусмотрена необязательная конструкция DEFAULT. В базе данных это значение применяется по умолчанию в тех случаях, если в

операторе INSERT не задано значение для такого столбца. Кроме прочих

значений, опция определения применяемого по умолчанию значения DEFAULT может включать литералы. Конструкции NОТ NULL, UNIQUE и CHECK рассматривались в предыдущем разделе. Остальные конструкции известны под названием ограничений таблицы и могут быть дополнительно обозначены с помощью следующей конструкции:

[PRIMARY KEY (column)

| FOREIGN KEY (columns) REFERENCES table (columns) [ON {UPDATE | DELETE} {CASCADE | SET DEFAULT | SET NULL |NO ACTION }] | UNIQUE (column) | CHECK (condition)

Эта конструкция позволяет в дальнейшем удалить ограничение, указав его имя в операторе ALTER TABLE, как описано ниже.

Конструкция PRIMARY KEY определяет один или несколько столбцов, которые образуют первичный ключ таблицы. По умолчанию для всех столбцов, представляющих первичный ключ, предусмотрено применение ограничения NOT NULL и UNIQUE. При создании таблицы разрешено использование только одной конструкции PRIMARY KEY. База данных отвергает все попытки выполнения операций INSERT или UPDATE, которые влекут за собой создание

строки с повторяющимся значением в столбце (столбцах) PRIMARY KEY.

Таким образом, в базе данных гарантируется уникальность значений

первичного ключа.

В конструкции FOREIGN KEY определяется внешний ключ (дочерней)

таблицы и ее связь с другой (родительской) таблицей. Эта конструкция

позволяет реализовать ограничения ссылочной целостности и состоит из

следующих частей:

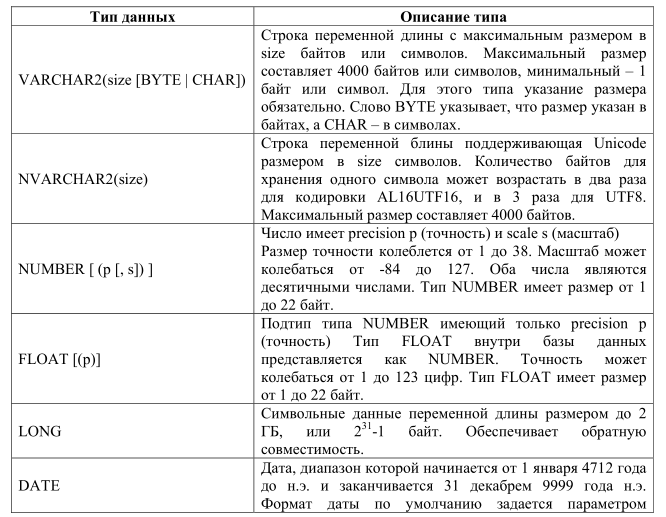
* Список столбцов columns, содержащий имена одного или нескольких столбцов создаваемой таблицы, которые образуют внешний ключ.
* Вспомогательная конструкция REFERENCES, указывающая на родительскую таблицу (т.е. таблицу, в которой определен соответствующий потенциальный ключ).
* Необязательное правило обновления (ON UPDATE) для определения взаимосвязи между таблицами, которое указывает, какое действие должно выполняться при обновлении в родительской таблице потенциального ключа, соответствующего внешнему ключу дочерней таблицы. В качестве параметра можно указать CASCADE, SET NULL, SET DEFAULT ИЛИ NO ACTION. Если конструкция ON UPDATE опущена, то по умолчанию подразумевается, что никакие действия не выполняются, в соответствии со значением NO ACTION.
* Необязательное правило удаления (ON DELETE) для определения взаимосвязи между таблицами, которое указывает, какое действие \должно выполняться при удалении строки из родительской таблицы, которая содержит потенциальный ключ, соответствующий внешнему ключу дочерней таблицы. Определение параметра совпадает с определением такого же параметра для правила ON UPDATE.

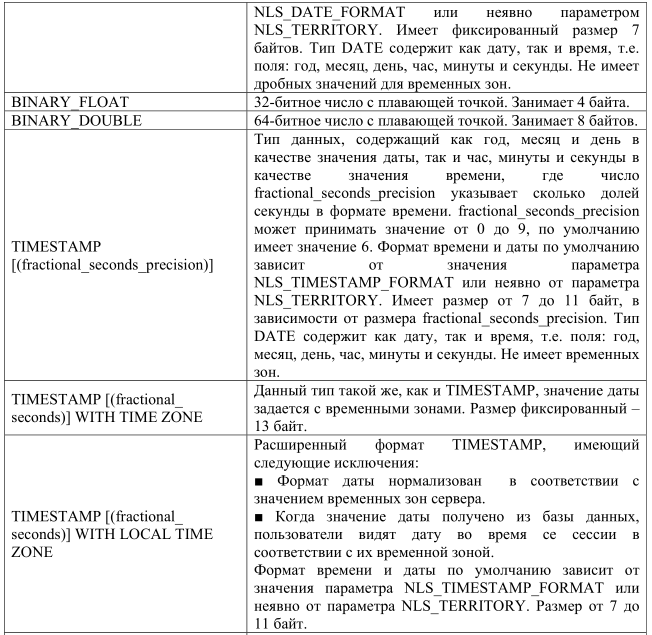
По умолчанию ограничение ссылочной целостности удовлетворяется,

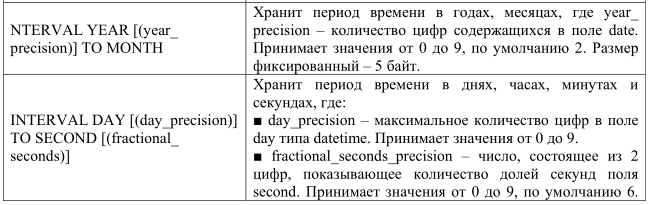
если любой компонент внешнего ключа имеет значение NULL или в родительской таблице есть соответствующая строка. В операторе создания таблицы может быть задано любое количество конструкций FOREIGN KEY.

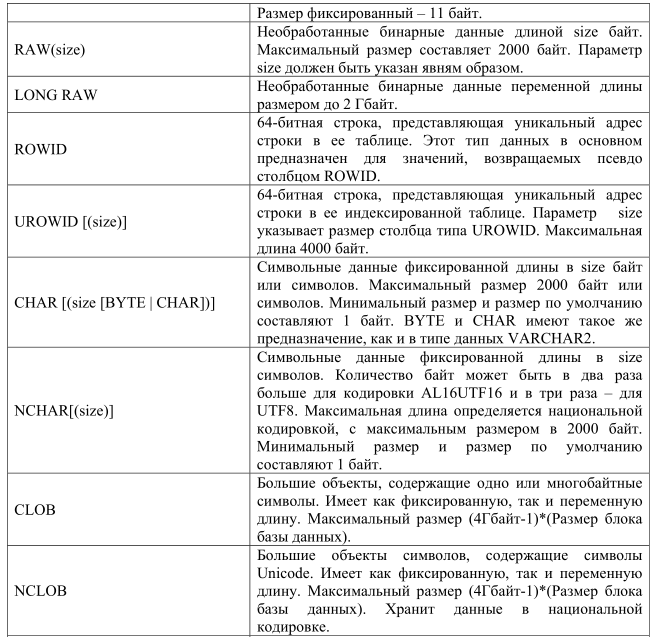
Конструкции CHECK и CONSTRAINT позволяют определять дополнительные ограничения. Если конструкция CHECK используется в качестве ограничения столбца, то она может ссылаться только на определяемый столбец. Ограничения фактически контролируются после применения каждого оператора SQL к таблице, на которой они заданы, но такая проверка может быть отложена до окончания той транзакции, в состав которой входит текущий оператор SQL.

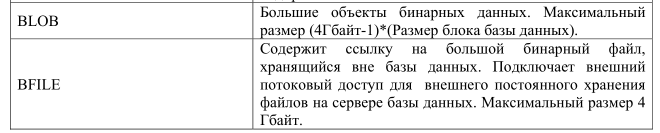
Типы данных ORACLE:











В примере демонстрируются широкие возможности представленной здесь версии оператора CREATE TABLE:

employee\_id NUMBER(6),

first\_name VARCHAR2(20),

last\_name VARCHAR2(25) CONSTRAINT emp\_last\_name\_nn NOT NULL,

email VARCHAR2(25) CONSTRAINT emp\_email\_nn NOT NULL,

phone\_number VARCHAR2(20),

hire\_date DATE CONSTRAINT emp\_hire\_date\_nn NOT NULL,

job\_id VARCHAR2(10) CONSTRAINT emp\_job\_nn NOT NULL,

salary NUMBER(8,2),

commission\_pct NUMBER(2,2),

manager\_id NUMBER(6),

department\_id NUMBER(4),

CONSTRAINT emp\_salary\_min CHECK (salary > 0),

CONSTRAINT emp\_email\_uk UNIQUE (email)

);

В ORACLE предусмотрено применение оператора ALTER TABLE для изменения структуры таблицы после ее создания. Определение оператора

ALTER TABLE состоит из шести опций, позволяющих выполнить следующие действия:

 переименовать таблицу;

 ввести новый столбец в таблицу;

 переименовать столбец таблицы;

 изменить тип данных столбца, значение по умолчанию и т.д.;

 удалить столбец из таблицы;

 ввести новое ограничение таблицы;

 удалить ограничение таблицы;

 перевод таблицы в режим «только для чтения».

Ниже приведен основной формат этого оператора.

ALTER TABLE table

{ RENAME {TO newtable | COLUMN column TO newcolumn } |

ADD { [COLUMN] columnname datatype [NOT NULL] [UNIQUE] [DEFAULT

expr [CHECK condition)] | CONSTRAINT [constraintname] ]

constraint\_definition] } |

MODIFY [COLUMN] columnname datatype

DROP { columnname | CONSTRAINT constraintname [RESTRICT | CASCADE];

Почти все параметры данного оператора совпадают с параметрами оператора CREATE TABLE, описанного в предыдущем разделе, В качестве параметра с определением ограничения таблицы constraintdefinition может применяться одна из конструкций PRIMARY KEY, UNIQUE, FOREIGN KEY или CHECK (см. выше). Конструкция ADD COLUMN аналогична конструкции определения столбца в операторе CREATE TABLE. В конструкции DROP COLUMN задается имя столбца, удаляемого из определения таблицы, и имеется необязательная опция, позволяющая указать, является ли действие операции DROP каскадным или нет, как показано ниже.

Пример: Изменение имени таблицы EMPLOYEE на EMP\_NEW:

ALTER TABLE EMPLOYEE RENAME TO EMP\_NEW;

Пример: Добавление к таблице EMPLOYEE столбца NEWCOL:

ALTER TABLE EMPLOYEE ADD (NEWCOL VARCHAR2 (100));

Пример: Изменение в таблице EMPLOYEE имени столбца NEWCOL на

RENCOL

ALTER TABLE EMPLOYEE RENAME COLUMN NEWCOL TO RENCOL;

Пример: Изменение в таблице EMPLOYEE типа данных столбца

NEWCOL на NUMBER.

ALTER TABLE EMPLOYEE MODIFY NEWCOL NUMBER;

Пример: Удаление из таблицы EMPLOYEE столбца NEWCOL.

ALTER TABLE EMPLOYEE DROP COLUMN NEWCOL;

Пример: Добавление к таблице EMPLOYEE ограничения первичного

ключа:

ALTER TABLE EMPLOYEE ADD PRIMARY KEY (EMPLOYEE\_ID);

Пример: Удаление ограничения первичного ключа из таблицы

EMPLOYEE.

ALTER TABLE EMPLOYEE DROP PRIMARY KEY;

Пример: Перевод таблицы EMPLOYEE в режим «только для чтения».

ALTER TABLE EMPLOYEE READ ONLY

С течением времени структура базы данных меняется: создаются новые

таблицы, а прежние становятся ненужными. Ненужные таблицы удаляются из базы данных с помощью оператора DROP TABLE, имеющего следующий

формат:

DROP TABLE tablename [RESTRICT | CASCADE];

Например, для удаления таблицы PropertyForRent можно использовать следующий оператор:

DROP TABLE PropertyForRent;

Однако следует отметить, что эта команда удалит не только указанную таблицу, но и все входящие в нее строки данных. Если требуется удалить из таблицы лишь строки данных, сохранив в базе описание самой таблицы, то следует использовать оператор DELETE. Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно:

 RESTRICT. Операция DROP отвергается, если в базе данных имеются

другие объекты, существование которых зависит от того, существует ли в базе данных удаляемая таблица.

 CASCADE. Операция DROP продолжается, и из базы данных

автоматически удаляются все зависимые объекты (и объекты, зависящие от этих объектов). Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может распространяться на значительную часть базы данных, поэтому подобные операторы следует использовать с максимальной осторожностью.

Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с неправильной структурой, можно воспользоваться оператором DROP TABLE

для ее удаления, после чего создать таблицу заново.

Создание индекса (оператор CREATE INDEX)

Индекс представляет собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы с учетом значений одного или нескольких ее столбцов. (Назначение индексов и способы их использования для повышения скорости выборки данных описаны в приложении В.) Наличие индекса может существенно повысить скорость выполнения некоторых запросов. Но поскольку индексы должны обновляться системой при каждом внесении изменений в их базовую таблицу, они создают дополнительную нагрузку на систему. Индексы обычно создаются с целью удовлетворения определенных критериев поиска, после того как таблица уже находилась некоторое время в работе и увеличилась в размерах. Создание индексов не предусмотрено стандартом языка SQL. Однако большинство диалектов поддерживает как минимум следующий оператор;

CREATE [UNIQUE] INDEX

ON Tablename (соlumn [{ASC | DESC ]})

Указанные в операторе столбцы составляют ключ индекса и должны быть перечислены в порядке уменьшения значимости. Индексы могут создаваться только для таблиц базы данных, но не для представлений. Если в операторе указано ключевое слово UNIQUE, уникальность значений ключа индекса будет автоматически установлено.

3 Создание простых запросов с использованием

операторов языка манипулирования данными

(SELECT, UPDATE, INSERT, DELETE).

**Краткие теоретические сведения**: Data Manipulation Language (DML) (язык манипулирования данными) это семейство опретаторов языка SQL, предназначенных для выборки и обновления данных. К данной группе относятся команды, предназначенные для работы с данными. Команды INSERT (вставка), UPDATE (обновление) и DELETE (удаление) работают с большинством строк таблицы, но работаю только с

одной таблицей одновременно. Команда SELECT (выбор) производит выбор данных из объектов баз данных, таких как таблицы, представления и могут использовать объекты типа функции.

Оператор INSERT используется для добавления строк в таблицу. Вы

можете указать следующую информацию при использовании оператора

INSERT:

 Таблица, в которую необходимо добавить строку.

 Список столбцов, для которых будут заданы значения.

 Список значений, которые будут храниться в указанных столбцах.

Во время добавления строки необходимо указать значения для первичного ключа и всех других столбцов, которые определены как NOT NULL. Нет необходимости указывать значения для остальных столбцов; им автоматически будет присвоено значение NULL. Эта форма оператора INSERT имеет следующий формат:

INSERT INTO TableName [(columnList)] VALUES (dataValueList)

Здесь параметр TableName может представлять имя таблицы базы данных.

Параметр colunmList представляет собой список, состоящий из имен одного или более столбцов, разделенных запятыми. Параметр coIumnLis является необязательным. Если он опущен, то предполагается использование списка из имен всех столбцов таблицы, указанных в том порядке, в котором они были описаны в операторе CREATE TABLE.

Пример использования команды INSERT без явного указания столбцов:

INSERT INTO Staff VALUES ('SG16', 'Alan', 'Brown', 'Assistant,

'M1', '25-MAY-1957', 8300, 'B003');

Пример с использованием значений по умолчанию:

INSERT INTO Staff (staffNo, fName, IName, position, salary,

branchNo) VALUES ('SG441' , 'Anne' , 'Jones', 'Assistant', 8100,

'BОО2' );

Оператор UPDATE позволяет изменять содержимое уже существующих строк указанной таблицы. Этот оператор имеет следующий формат:

UPDATE TableName SET columnNamel = dataValuel [, columnName2 =dataValue2] [WHERE searchCondition];

Здесь параметр tableName представляет имя таблицы базы данных. В конструкции SET указываются имена одного или более столбцов, данные в которых необходимо изменить. Конструкция WHERE является необязательной.

Если она опущена, значения указанных столбцов будут изменены во всех строках таблицы. Если конструкция WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию поиска, заданному в параметре searchCondition. Параметры dataValuel, dataValue2t... представляют новые значения соответствующих столбцов и должны быть совместимы с ними по типу данных.

Пример: Обновление всех строк таблицы с помощью оператора UPDATE.

UPDATE Staff SET salary = salary\*0.03;

Поскольку изменения касаются всех строк таблицы Staff, конструкцию WHERE указывать не требуется.

Пример: Обновление некоторых строк таблицы с помощью оператора

UPDATE

UPDATE Staff SET salary = salary\*0.05 WHERE position = 'Manager';

Оператор DELETE позволяет удалять строки данных из указанной

таблицы. Этот оператор имеет следующий формат:

DELETE FROM tableName [WHERE searchCondition;

Как и в случае операторов INSERT и UPDATE, параметр tableName представляет собой имя таблицы базы данных. Параметр searchCondition является необязательным — если он опущен, из таблицы будут удалены все существующие в ней строки. Однако сама по себе таблица удалена не будет. Если конструкция WHERE присутствует, из таблицы будут удалены только тестроки, которые удовлетворяют условию отбора, заданному параметром searchCondition.

Пример: Удаление определенных строк таблицы (оператор DELETE).

DELETE FROM Viewing WHERE propertyNo = 'PG4';

Назначение оператора SELECT состоит в выборке и отображении данных одной или более таблиц базы данных. Это исключительно мощный оператор, способный выполнять действия, эквивалентные операторам реляционной алгебры выборки, проекции и соединения, причем в пределах единственной выполняемой команды. Оператор SELECT является чаще всего используемой командой языка SQL. Общий формат оператора SELECT имеет следующий вид:

SELECT [DISTINCT | ALL] { \* | [columnExpression AS newName] [ , '

. . . '] }

FROM TableName [alias] [ , . . - ]

[WHERE condition]

[GROUP BY columnwise]

[HAVING condition]

ORDER BY columnList];

Здесь параметр columnExpression представляет собой имя столбца или выражение из нескольких имен. Параметр TableName является именем существующей в базе данных таблицы (или представления), к которой необходимо получить доступ. Необязательный параметр alias — это сокращение, устанавливаемое для имени таблицы TableName. Обработка элементов оператора SELECT выполняется в следующей последовательности:

 FROM. Определяются имена используемой таблицы или нескольких

таблиц.

 WHERE. Выполняется фильтрация строк объекта в соответствии с

заданными условиями.

 GROUP BY. Образуются группы строк, имеющих одно и то же

значение в указанном столбце.

 HAVING. Фильтруются группы строк объекта в соответствии с

указанным условием. SELECT. Устанавливается, какие столбцы должны

присутствовать в выходных данных.

 ORDER BY. Определяется упорядоченность результатов выполнения

оператора.

Порядок конструкций в операторе SELECT не может быть изменен. Только две конструкции оператора – SELECT и FROM – являются обязательными, все остальные конструкции могут быть опущены. Операция выборки с помощью оператора SELECT является замкнутой, в том смысле, что результат запроса к таблице также представляет собой таблицу. Существует множество вариантов использования данного оператора, что иллюстрируется приведенными ниже примерами:

SELECT staffNo, fName, IName, position, sex, DOB, salary, branchNo

FROM Staff;

Поскольку выборка всех имеющихся в таблице столбцов выполнятся достаточно часто, в языке SQL определен упрощенный вариант записи значения «все столбцы» – вместо имен столбцов указывается символ звездочки (\*).

В этом примере на основе таблицы Staff создается новая таблица,

включающая только указанные в запросе столбцы staffNo, fName, IName и salary, причем именно в этом порядке:

SELECT staffNo, fName, IName, salary FROM Staff;

В приведенных выше примерах в результате выполнения операторов SELECT выбирались все строки указанной таблицы. Однако очень часто требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе конструкции WHERE. Она состоит из ключевого слова WHERE, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска (или предикатов, если пользоваться терминологией ISO).

 Сравнение. Сравниваются результаты вычисления одного выражения с

результатами вычисления другого выражения.

 Диапазон. Проверяется, попадает ли результат вычисления выражения

в заданный диапазон значений.

 Принадлежность к множеству. Проверяется, принадлежит ли

результат вычисления выражения к заданному множеству значений.

 Соответствие шаблону. Проверяется, отвечает ли некоторое строковое

значение заданному шаблону.

 Значение NULL. Проверяется, содержит ли данный столбец NULL

(неопределенное значение).

Например:

SELECT staffNo, fName, IName, position, salary FROM Staff WHERE

salary > 10000;

В этом запросе используются таблица Staff и предикат salary > 10000. При выполнении запроса будет создана новая таблица, содержащая только те строки таблицы Staff, в которых значение столбца salary больше 10 000 фунтов стерлингов.

Более сложные предикаты могут быть построены с помощью логических операций AND, OR или NOT, а также с помощью скобок, используемых для определения порядка вычисления выражения (если это необходимо или желательно).

Вычисление выражений в условиях выполняется по следующим

правилам.

 Выражение вычисляется слева направо.

 Первыми вычисляются подвыражения в скобках.

 Операции NOT выполняются перед операциями AND и OR.

 Операции AND выполняются перед операциями OR.

Для устранения любой возможной неоднозначности рекомендуется

использовать круглые скобки.

Пример: Сложные условия поиска

SELECT \* FROM Branch WHERE city = 'London' OR city = 'Glasgow';

В этом примере для выборки сведений об отделениях компании,

находящихся в Лондоне (city = 'London') илм Глазго (city = 'Glasgow'), в

конструкции WHERE используется логический оператор OR.

В общем случае строки в результирующей таблице запроса SQL не

упорядочены каким-либо определенным образом (хотя в некоторых СУБД может быть предусмотрено применение по умолчанию определенного способа упорядочения, например по первичному ключу). Однако их можно отсортировать надлежащим образом, для чего в оператор SELECT помещается конструкция ORDER BY. Конструкция ORDER BY включает список разделенных запятыми идентификаторов столбцов, по которым требуется упорядочить результирующую таблицу запроса. Идентификатор столбца может представлять собой либо его имя, либо номер, который обозначает элемент списка SELECT в соответствии с его позицией в этом списке. Самый левый элемент списка имеет номер 1, следующий — 2 и т.д. Номера столбцов могут использоваться в тех случаях, когда столбцы, по которым следует упорядочить результат, являются вычисляемыми, а конструкция AS с указанием имени этого столбца в операторе SELECT отсутствует. Конструкция ORDER BY позволяет упорядочить выбранные записи в порядке возрастания (ASC) или убывания (DESC) значений любого столбца или комбинации столбцов. Номера столбцов

являются устаревшим средством стандарта ISO, поэтому их не рекомендуется использовать. Однако в некоторых диалектах SQL требуется, чтобы конструкция ORDER BY обязательно присутствовала в списке выборки оператора SELECT. В любом случае конструкция ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример: Сортировка по значениям одного столбца

SELECT staffNo, fName, IName, salary FROM Staff ORDER BY salary

DESC;

Стандарт ISO содержит определение следующих пяти агрегирующих функций:

 COUNT — возвращает количество значений в указанном столбце;

 SUM — возвращает сумму значений в указанном столбце;

 AVG — возвращает усредненное значение в указанном столбце;

 MIN — возвращает минимальное значение в указанном столбце;

 МАХ — возвращает максимальное значение в указанном столбце.

Все эти функции оперируют со значениями в единственном столбце

таблицы и возвращают единственное значение. Функции COUNT, MIN и МАХ применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей. За исключением COUNT ( \* ) , при вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся непустым значениям столбца. Вариант COUNT (\*) является особым случаем использования функции COUNT — его назначение состоит в подсчете всех строк в таблице, независимо от того, содержатся там пустые, повторяющиеся или любые другие значения.

Например: Вычислить значение минимальной, максимальной и средней заработной платы:

SELECT MIN(salary) AS min, MAX(salary) AS max, AVG(salary) AS avg

FROM Staff;

В этом примере необходимо обработать сведения обо всем персонале компании, поэтому использовать конструкцию WHERE не требуется. Необходимые значения могут быть вычислены с помощью функций MIN, MAX и AVG, применяемых к столбцу salary таблицы Staff. обычно размещаемым в конце отчетов. В итогах все детальные данные отчета сжимаются в одну обобщающую строку. Однако очень часто в отчетах требуется формировать и промежуточные итоги. Для этой цели в операторе SELECT может указываться конструкция GROUP BY. Запрос, в котором присутствует конструкция GROUP BY, называется группирующим запросом, поскольку в нем группируются

данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная итоговая строка.

Столбцы, перечисленные в конструкции GROUP BY, называются

группируемыми столбцами. Стандарт ISO требует, чтобы конструкции SELECT и GROUP BY были тесно связаны между собой. При использовании в операторе

SELECT конструкции GROUP BY каждый элемент списка в списке выборки

SELECT должен иметь единственное значение для всей группы. Более того, конструкция SELECT может включать только следующие типы элементов:

 имена столбцов;

 агрегирующие функции;

 константы;

 выражения, включающие комбинации перечисленных выше

элементов.

Все имена столбцов, приведенные в списке выборки SELECT, должны присутствовать и в конструкции GROUP BY, за исключением случаев, когда имя столбца используется только в агрегирующей функции. Противоположное утверждение не всегда справедливо – в конструкции GROUP BY могут присутствовать имена столбцов, отсутствующие в списке выборки SELECT.

Если совместно с конструкцией GROUP BY используется конструкция WHERE, то она обрабатывается в первую очередь, а группированию подвергаются только те строки, которые удовлетворяют условию поиска. Стандартом ISO определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значения NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Пример использования конструкции GROUP BY:

SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum FROM

Staff GROUP BY branchNo ORDER BY branchNo;

Внешний (второй) оператор SELECT использует результат выполнения

внутреннего (первого) оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут находиться в конструкциях WHERE и HAVING внешнего оператора SELECT – в этом случае они получают название подзапросов, или вложенных запросов. Кроме того, внутренние операторы SELECT могут использоваться в операторах INSERT, UPDATE и DELETE. Существуют три типа подзапросов.

1. Скалярный подзапрос возвращает значение, выбираемое из

пересечения одного столбца с одной строкой, т.е. единственное значение. В принципе скалярный подзапрос может использоваться везде, где требуется указать единственное значение.

2. Строковый подзапрос возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Строковый подзапрос может использоваться везде, где применяется конструктор строковых значений.

3. Табличный подзапрос возвращает значения одного или нескольких

столбцов таблицы, размещенные в более чем одной строке.

Табличный подзапрос может использоваться везде, где допускается

указывать таблицу, например как операнд предиката IN.

Пример: Использование подзапроса с проверкой на равенство.

SELECT staffNo, fName, IName, position FROM Staff WHERE branchNo =( SELECT branchNo FROM Branch WHERE street = '163 Main St' );

К подзапросам применяются следующие правила и ограничения.

1. В подзапросах не должна использоваться конструкция ORDER BY,

хотя она может присутствовать во внешнем операторе SELECT.

2. Список выборки SELECT подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе используется ключевое слово EXISTS.

3. По умолчанию имена столбцов в подзапросе относятся к таблице,

имя которой указано в конструкции FROM подзапроса. Однако

разрешается ссылаться и на столбцы таблицы, указанной в

конструкции FROM внешнего запроса, для чего используются

уточненные имена столбцов.

4. Если подзапрос является одним из двух операндов, участвующих в

операции сравнения, то подзапрос должен указываться в правой

части этой операции. Например, приведенный ниже вариант записи

запроса из предыдущего примера является некорректным,

поскольку подзапрос размещен в левой части операции сравнения

со значением столбца salary.

Все рассмотренные выше примеры имеют одно и то же важное

ограничение: помещаемые в результирующую таблицу столбцы всегда

выбираются из единственной таблицы. Однако во многих случаях этого оказывается недостаточно. Для того чтобы объединить в результирующей таблице столбцы из нескольких исходных таблиц, необходимо выполнить операцию соединения.

В языке SQL операция соединения используется для объединения

информации из двух таблиц посредством образования пар связанных строк, выбранных из каждой таблицы. Помещаемые в объединенную таблицу пары строк составляются по равенству входящих в них значений указанных столбцов.

Если необходимо получить информацию из нескольких таблиц, то можно либо применить подзапрос, либо выполнить соединение таблиц. Если результирующая таблица запроса должна содержать столбцы из разных исходных таблиц, то целесообразно использовать механизм соединения таблиц. Для выполнения соединения достаточно в конструкции FROM указать имена двух и более таблиц, разделив их запятыми, после чего включить в запрос конструкцию WHERE с определением столбцов, используемых для соединения указанных таблиц. Помимо этого, вместо имен таблиц можно использовать псевдонимы, назначенные им в конструкции FROM. В этом случае имена таблиц и назначаемые им псевдонимы должны разделяться пробелами. Псевдонимы могут использоваться с целью уточнения имен столбцов во всех тех случаях, когда возможна неоднозначность в отношении того, к какой таблице относится тот или иной столбец. Кроме того, псевдонимы могут использоваться для сокращенного обозначения имен таблиц. Если для таблицы определен псевдоним, он может применяться в любом месте, где требуется указание имени этой таблицы.

Пример: Простое соединение.

SELECT c.clientNo, fName, IName, propertyNo, comment FROM Client c,

Viewing v WHERE c.clientNo = v.clientNo;

Стандарт SQL дополнительно предоставляет следующие способы

определения данного соединения:

FROM Client с JOIN Viewing v ON с.clientNo = v.clientNo

FROM Client JOIN Viewing USING clientNo

FROM Client NATURAL JOIN Viewing

Используем левое внешнее соединение этих двух таблиц, которое

выглядит следующим образом:

SELECT b.\*, р.\* FROM Branch b LEFT JOIN PropertyForRent p ON

b.bCity = p.pCity;

В этом примере за счет применения левого внешнего соединения в результирующую таблицу попали не только две строки, в которых имеется соответствие между названиями городов, но также та строка первой из соединяемых таблиц (левой), которая не нашла себе соответствия во второй таблице (правой). В этой строке все поля второй таблицы заполнены значениями NULL.

Пример: Правое внешнее соединение.

Перечислите отделения компании и сдаваемые в аренду объекты, которые расположены в одном и том же городе, а также все остальные объекты собственности, не удовлетворяющие условию запроса.

Используем правое внешнее соединение этих двух таблиц, которое

выглядит следующим образом:

SELECT b.\* , p.\* FROM Branch b RIGHT JOIN PropertyForRent p ON

b.bCity = p.pCity;

Используем полное внешнее соединение этих таблиц, которое выглядит

следующим образом:

SELECT b.\*, р.\* FROM Branchl b FULL JOIN PropertyForRent p ON

b.bCity = p.pCity;

В языке SQL можно использовать обычные операции над множествами – объединение (union), пересечение (intersection) и разность (difference), –позволяющие комбинировать результаты выполнения двух и более запросов в единую результирующую таблицу.

Объединением двух таблиц А и В называется таблица, содержащая все строки, которые имеются в первой таблице (А), во второй таблице (В) или в обеих этих таблицах одновременно.

Пересечением, двух таблиц называется таблица, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.

Разностью двух таблиц А и В называется таблица, содержащая все

строки, которые присутствуют в таблице А, но отсутствуют в таблице В. На таблицы, которые могут комбинироваться с помощью операций над множествами, накладываются определенные ограничения. Самое важное из них состоит в том, что таблицы должны быть совместимы, по соединению – т.е. они должны иметь одну и ту же структуру. Это означает, что таблицы должны иметь одинаковое количество столбцов, причем в соответствующих столбцах должны размещаться данные одного и того же типа и длины. Обязанность убедиться в том, что значения данных соответствующих столбцов принадлежат одному и тому же домену, возлагается на пользователя. Три операции над множествами, предусмотренные стандартом ISO, носят название UNION, INTERSECT и EXCEPT. В каждом случае формат конструкции с операцией над множествами должен быть следующим:

Создайте список всех регионов, в которых либо находится отделение

компании, либо располагаются сдаваемые в аренду объекты.

(SELECT city или (SELECT \* FROM Branch FROM Branch WHERE city IS

NOT NULL) WHERE city IS NOT NULL)

UNION

(SELECT city (SELECT \* FROM PropertyForRent FROM PropertyForRent

WHERE city IS NOT NULL) WHERE city IS NOT NULL);

Этот запрос выполняется посредством подготовки результирующей таблицы первого запроса и результирующей таблицы второго запроса с последующим слиянием обеих таблиц в единую результирующую таблицу, которая включает все строки из обеих промежуточных таблиц, но с удалением повторяющихся строк.

(SELECT city или (SELECT \* FROM Branch) FROM Branch)

INTERSECT

(SELECT city или (SELECT \* FROM PropertyForRent) FROM

PropertyForRent);

Этот запрос выполняется посредством подготовки результирующей таблицы первого запроса и результирующей таблицы второго запроса с последующим созданием единой результирующей таблицы, включающей только те строки, которые являются общими для обеих промежуточных таблиц.



Рис 1.2 Пример реляционной модели.

ВАРИАНТЫ ЗАДАНИЙ (по последним цифрам зачетной книжки или по их сумме).

1. Разработать базу данных для выдачи банковского кредита.
2. Учет результатов сдачи вступительных экзаменов
3. Учет вкладов в банке
4. Добровольное медицинское страхование
5. Штатное расписание
6. Учет затрат на изготовление изделий
7. Учет эксплуатации транспортных средств
8. Учет подписки на печатные издания
9. Туристическое агентство
10. Учет нарушений правил дорожного движения
11. Учет сделок с недвижимостью
12. Учет выполнения заказов
13. Оптовый магазин. Учет продаж товара
14. Прокат автомобилей
15. Учет выполнения работ
16. Платный прием в поликлинике
17. Книжный магазин
18. Чрезвычайные ситуации
19. Субсидии
20. Отдел кадров
21. Реализация товаров
22. Библиотека
23. Автовокзал
24. Расчеты оплаты за тепло
25. Оздоровительный комплекс
26. Гостиница
27. Учёт прихода товаров на почтамт
28. АРМ бухгалтера. Формирование расчётных листов сотрудников
29. Экспедитор (Заказы товаров магазинами)
30. Рекламное агентство
31. Платежи в банке
32. Оплата за квартиру
33. Фирма металлопластиковых окон
34. Контроль оплаты за обучение
35. Услуги интернет-провайдера
36. Кабельное телевидение. Учёт клиентов
37. Сервисный центр по обслуживанию мобильных телефонов

1) Для разработанной модели построить физическую модель базы данных с указанием названий таблиц, полей, первичных, вторичных ключей индексов и ограничений. Определить самостоятельно типы полей в таблицах. Ключевые поля таблиц выделить жирным шрифтом.

2) Написать 5 различных SQL запросов обеспечивающих выборку данных из таблиц.

Полученные скрипты создания таблиц и SQL запросы выложить в эссе.

(Выкладывается только текст в формате SQL, **диаграмму сущность-связь приводить не нужно**. Также необходимо наличие комментариев поясняющих работу SQL кода.)