1. Реляционные базы данных

Реляционная база данных (БД) – это такое средство хранения и доступа к данным, которое позволяет конечному пользователю не учитывать все тонкости процессов размещения и обработки данных компьютером.

Отношение – двумерная структура данных, которая в БД обычно представлена в виде таблицы. Атрибуты, или столбцы, содержат информацию в этой структуре. Значения данных о конкретном объекте хранятся в кортеже, или в строке таблицы. Между двумя или несколькими таблицами могут быть определены связи. Для однозначного определения строки таблицы используют первичный ключ – атрибут (или группа атрибутов), который содержит уникальны данные (не допускаются NULL-значения). Таблица имеет только один первичный ключ.

Связь между таблицами устанавливается при помощи общего атрибута.

PRIMARY KEY	**Не позволяет значениям столбца или набора столбцов дублироваться и быть неопределенными.**
FOREIGN KEY	**Требует, чтобы каждое значение столбца или набора столбцов в одной таблице, соответствовало значению столбца или набора столбцов, объявленных в другой с правилом целостности UNIQUE или PRIMARY KEY (правила целостности FOREIGN KEY также задают действия ссылочной целостности, которые нужно выполнять над зависимыми данными при изменении данных, на которые они ссылаются).**

В каждом отделе работает несколько работников.

Таблица DEPT

DNUM	DNAME	LOCATION
10	SALES	LONDON
20	PROJECTS	LONDON

(PK)

Таблица EMP

ENUM	ENAME	SALARY	DNUM
1001	TOMM	15000	10
1002	KATE	12000	20
1003	BILL	12000	20

(PK) **(FK)**

Каждый работник может работать на нескольких работах.

Таблица EMP

ENUM	ENAME	PROFESSION
1001	TOMM	DESIGNER
1002	KATE	DEVELOPER
1003	BILL	DEVELOPER

(PK)

Таблица FIRM

FNUM	FNAME	SALARY	ENUM
201	SKY_STAR	15000	1001
202	SOFT_MOBIL	12000	1001
203	ASTRA_AG	22000	1002
204	SOFT_CAR	25000	1003
205	CITY_SYSTEMS	19000	1003

(PK) **(FK)**

1. Команды манипулирования данными

1.1. Команда SELECT

SELECT [{ ALL | DISTINCT }] ***список выбора***
FROM ***список таблиц*** **|** ***список представлений***
WHERE ***условие***
START WITH ***условие***
CONNECT BY ***условие***
GROUP BY ***поле*** **[,] [HAVING** ***условие*** **]**
UNION ***команда select***
UNION ALL
INTERSECT
MINUS
FOR UPDATE [OF ***поле*** **[,] [NOWAIT]]**
ORDER BY ***поле*** **|** ***позиция*** **[ASC | DESC]**

1.1.1. Использование псевдостолбцов

Псевдостолбец	Возвращаемое значение
sequence.CURRVAL	Последнее значение последовательности.
sequence.NEXTVAL	Следующее значение последовательности.
LEVEL	Глубина запроса внутри дерева.
ROWID	Точное расположение строки данных в памяти.
ROWNUM	Порядковый номер выбранной строки
SYSDATE	Текущая дата и время
UID	Уникальный идентификатор текущего пользователя
USER	Имя, под которым пользователь зарегистрировался в базе данных.

1.2. Команда INSERT

INSERT INTO [***схема*****.] {** ***таблица*** **|** ***представление*** **}**
[***@связь с базой данных*****] (***поле*** **[,])**
{ VALUES (***выражение*** **[,]) |** ***запрос*** **}**

```
INSERT INTO emp (empno, ename)
    VALUES (7777, 'BILL');

INSERT INTO emp_copy
    SELECT * FROM emp;
```

1.3. Команда UPDATE

UPDATE [*схема.*] { *таблица* | *представление* }
[@*связь с базой данных*] [*алиас*]
SET { *поле* = { *выражение* | *запрос* } [,] | (*поле* [,]) = *запрос* }
[WHERE *условие*]

```
UPDATE emp  SET sal =sal * 1.1,
      WHERE job = 'CLERK';

UPDATE emp_copy ec    SET (ename, job, mgr)
      = (SELECT e.ename, e.job, e.mgr FROM emp e
            WHERE e.empno = ec.empno)
      WHERE ename LIKE 'A%';
```

1.4. Команда DELETE

DELETE [FROM] [*схема.*] { *таблица* | *представление* }
[@*связь с базой данных*] [*алиас*]
[WHERE *условие*]

2. Команды определения данных

Таблицы

CREATE TABLE [схема.] *имя_таблицы*
(*имя_столбца тип_данных* [DEAFAULT *выражение*]
[*ограничение_столбца*] [,])
AS запрос

```
CREATE TABLE dept
    (dnum     number(2)     primary key,
     dname    varchar2(20),
     location varchar2(20)
    );
```

Представления

CREATE [OR REPLACE] VIEW [*схема*.] *имя_представления*
AS select-*команда*
[WITH READ ONLY | WITH CHECK OPTION]

```
CREATE VIEW emp_cnt_firm AS
SELECT e.ename, count(f.fname) cnt_firm, sum(salary) sum_sl
  FROM emp e, firm f     WHERE e.enum=f.enum
  GROUP BY e.ename HAVING count(f.fname) > 1;

ENUM     CNT_FIRM   SUM_SL
-------------------------------------------------
TOMM          2          27000
 BILL         2          44000
```

3. Команды управления транзакциями

3.1. Понятие транзакции

ТРАНЗАКЦИЯ - это логическая единица работы, составленная из одной или нескольких команд SQL

Банковская транзакция

Транзакция начинается
Уменьшить накопительный счет
UPDATE savings_accounts SET balance = balance - 500
WHERE account = 3209;

Увеличить чековый счет
UPDATE checking_accounts SET balance = balance + 500
WHERE account = 3208;

Записать в журнале транзакций
INSERT INTO journal
VALUES(journal_seq.NEXTVAL,'1B',3209, 3208, 500);

Закончить транзакцию
COMMIT WORK;

Транзакция заканчивается

3.2. ORACLE и управление транзакциями

Транзакция в ORACLE начинается, когда встречается первая выполнимая команда SQL.

Транзакция заканчивается, когда происходит одно из следующих событий:

- **выдана команда COMMIT**
- **выдана команда ROLLBACK (без фразы SAVEPOINT)**
- **выдана команда DDL (CREATE, DROP, RENAME, ALTER, …).**
- **пользователь отсоединяется от ORACLE. (транзакция подтверждается.)**
- **имеет место аварийное прекращение пользовательского процесса. (транзакция откатывается.)**

После окончания одной транзакции очередная выполнимая команда SQL автоматически начинает следующую транзакцию.

3.3. Подтверждение транзакций

COMMIT WORK;
COMMIT;

3.4. Откат транзакций

ROLLBACK WORK;
ROLLBACK;

ROLLBACK TO SAVEPOINT *имя_точки_сохранения*;
ROLLBACK TO *имя_точки_сохранения*;

3.5. Точки сохранения

SAVEPOINT *имя_точки_сохранения*;

Выборка данных в PL/SQL

Атрибуты курсора

Наименование	Возвращаемое значение
%FOUND	TRUE, если успешно выбрана хотя бы одна строка
%NOTFOUND	TRUE, если инструкция не выбрала ни одной строки
%ROWCOUNT	Количество строк, выбранных из курсора на данный момент
%ISOPEN	TRUE, если курсор открыт
%BULK_ROWCOUNT	Коллекция, в которой для каждого элемента исходной коллекции оператора FORALL указано количество строк, обработанных SQL-инструкцией
%BULK_EXCEPTIONS	Коллекция, в которой для каждого элемента исходной коллекции оператора FORALL, вызвавшего программную ошибку, указано инициированное исключение

1. Неявные курсоры

PL/SQL автоматически создает неявный курсор для каждой выполняемой инструкции DML (INSERT, UPDATE, DELETE) или инструкции SELECT INTO. Курсор называется неявным, т.к. Oracle автоматически выполняет многие связанные с ним операции. Для обращения к такому курсору используется ключевое слов SQL.

Неявный курсор создается для инструкции SELECT, которая

- **определяется в исполняемом разделе, а не в разделе объявлений**
- **содержит предложение INTO, которое является частью языка PL/SQL, а не SQL**
- **не требует открытия, выборки данных и закрытия.**

SELECT *список_столбцов* [BULK COLLECT] INTO *список_переменных*
. . . оставшаяся часть инструкции SELECT

```
DECLARE
        l_sal   sal%ROWTYPE;
BEGIN
        SELECT  *  INTO  l_sal  FROM  sal  WHERE  snum = 1001;
END;
```

Инструкция SELECT INTO (без BULK COLLECT) должна возвращать одну строку. Oracle инициирует исключение, если

- **по запросу не найдено ни одной строки – NO_DATA_FOUND**
- **возвращено несколько строк – TOO_MANY_ROWS**

2. Явные курсоры

Явный курсор – это инструкция SELECT, явно определенная в разделе объявлений как курсор. При объявлении курсору присваивается имя.
Для INSERT, UPDATE и DELETE явные курсоры создавать нельзя.

2.1. Объявление явного курсора

CURSOR ***имя_курсора*** **[(*параметр* [, *параметр* …])]**
[RETURN *спецификация_return*]
IS ***инструкция_SELECT***
[FOR UPDATE [OF *список_столбцов*]];

- **Курсор без параметров**

```
CURSOR c_ord IS
    SELECT odate FROM ord;
```

2.2. Открытие явного курсора

OPEN ***имя_курсора*** **[(*аргумент* [, *аргумент* …])];**

Оператор OPEN выполняет содержащийся в курсоре запрос. Он формирует результирующий набор строк, но не извлекает данные – это делает оператор FETCH.

```
IF NOT c_sal%ISOPEN THEN
   OPEN c_sal;
END IF;
```

2.3. Выборка данных из явного курсора

FETCH ***имя_курсора*** **INTO** ***запись_или_список_переменных*****;**

- **Выборка данных из курсора в запись**

```
DECLARE
    CURSOR c_ord IS
       SELECT * FROM ord;
    ord_rec   ord%ROWTYPE;
BEGIN
   OPEN c_ord;
   FETCH c_ord INTO ord_rec;
```

2.4. Закрытие явного курсора

CLOSE ***имя_курсора*****;**

Обработка ошибок

```
DECLARE
    (объявления переменных)
BEGIN
    (главная обрабатывающая часть программы)
EXCEPTION
    WHEN no_data_found THEN
        out_status := 'Data not …..';
        return_code := 5;
    WHEN too_many_rows THEN
        out_status := 'Query …..';
        return_code := 6;
END;
```

1. внутренние исключения,
2. исключения, предусмотренные пользователем.

- NO_DATA_FOUND
- STORAGE_ERROR
- TIMEOUT_ON_RESOURCE
- TOO_MANY_ROWS
- ZERO_DIVIDE

OTHERS – обработка всех (остальных) ошибок

SQLCODE – код ошибки
SQLERRM – описание ошибки

INSERT INTO errors VALUES (SQLCODE, SQLERRM); -- незаконно

```
DECLARE
    err_num NUMBER;
    err_msg CHAR(100);
BEGIN
    …
EXCEPTION
    WHEN OTHERS THEN
        err_num := SQLCODE;
        err_msg := SUBSTR(SQLERRM, 1, 100);
        INSERT INTO errors VALUES (err_num, err_msg);
END;
```

Спецификация пакета

```
CREATE OR REPLACE PACKAGE my_pack IS
    PROCEDURE fill;
    PROCEDURE del;
    PROCEDURE upd_10 (p_firm varchar2);
    PROCEDURE show (p_firm varchar2);
END lib;
/
```

Тело пакета

```
CREATE OR REPLACE PACKAGE BODY my_pack IS
    PROCEDURE fill  IS
    BEGIN
     INSERT INTO emp VALUES(1001,'Tomm','designer');
     . . .
     . . .
     INSERT INTO firm VALUES(201,'Sky_Star',15000,1001);
     . . .
     . . .
     COMMIT;
    EXCEPTION
     WHEN others THEN
     . . .
    END;
    PROCEDURE del
     . . .
    END;
    PROCEDURE upd_10 (p_firm varchar2)
     . . .
    END;
    PROCEDURE show (p_firm varchar2)
     . . .
    END;
END lib;
/
```