CHAPTER 3

Oracle SQL and iSQL*Plus

3.1 SQL

SQL is an industry standard language for querying databases. The benefits of standardisation in information systems are widely recognised, and include re-usablility and portability of code, improved communication and training, and the ability to create cross-platform applications.

3.2 SQL Standards

Unfortunately the situation regarding "standard" SQL is not straightforward. The standard has evolved over the years in line with increasingly powerful DBMS capabilities and improved understanding of the field, and several versions are now in use. The most commonly accepted version is SQL-89, which is a close equivalent to Entry-level SQL-92, and will be referred to in this document as "ANSI-standard SQL".

SQL:1999 adds facilities for the specification and manipulation of object-relational databases to those of ANSI-standard SQL, and will be discussed later in the module.

Few DBMSs adhere strictly to the ANSI standard. In particular, suppliers tend to add their own extensions and enhancements, and Oracle is no exception. However, it is possible to use only those parts of Oracle SQL that adhere to the ANSI standard. We will not attempt to do so in this module.

It is safe to say that studying Oracle SQL will provide the student with a good understanding of any other version they will meet, despite the minor variations. For the remainder of this document the term SQL will refer to Oracle SQL (unless otherwise stated).

3.3 iSQL*Plus

As discussed in the previous chapter, SQL has two components. The Database Definition Language (DDL) provides the means to create and manipulate the structure of a database. The Data Manipulation Language (DML) enables the data held in the database to be manipulated. They can be said to be concerned respectively with the data container and the data contained.

There are many tasks that users at all levels need to, when interacting with a database, that do not belong to either of these categories. For example the database administrator will want to create new user accounts, and to grant users permission to access or change different parts of the database. Individual users may want to save queries they have written so that they can be re-used, and to output the results of running queries to text files.

All DMBSs must provide the means for such tasks to be accomplished. Many (including Oracle) provide graphical tools for such purposes. We are going to use iSQL*Plus for these purposes.

3.4 Logging on to Oracle

When you connect to iSQL*Plus you will be asked for your name, password and host string. The host string is "acal".

3.5 Entering SQL Statements and iSQL*Plus Commands

Entering these details will cause the Oracle iSQL*Plus window to open, a connection to the Oracle "acal" database will be made. SQL queries and statements are simply entered in the window. Click on the 'Execute' button will cause them to be executed, and the results will be displayed.

iSQL*Plus commands are entered in the same way. Click on link, Help, will cause the system to display a list of the iSQL*Plus commands. Click on the link of one of the commands will display the description of the purpose of the command, and how it is invoked. Alternatively, typing HELP followed by the name of one of these commands will display a brief description.

Note that Oracle is not by default case sensitive, so you do not have to be consistent with your use of upper and lower case letters when entering iSQL*Plus commands and SQL statements. Table names, attribute names, etc. are converted to upper case by Oracle when they are stored. However, your code will be more readable if you are consistent. *Note that this does not apply to data, which will be stored in the case that it was entered.* Oracle can be forced to be case sensitive, but the examples and exercises in this booklet will accept the default behaviour.

3.6 The SQL Data Definition Language

In this section we will consider two statements that form part of the DDL of SQL. This will enable us to create database tables, so that we can populate them, and manipulate the data they contain. We will return in a later section to take a more detailed look at these and other DDL statements.

CREATE - used to create table definitions

DROP - used to remove unwanted tables from the database.

In a live database, i.e. a database supporting some aspect of an organisation's operation, the DDL would be used by the *Database Administrator* (DBA), an individual responsible for establishing and maintaining the definition of the database tables supporting the organisation's information needs.

3.6.1 Creating Tables

In its simplest form the SQL statement used to create a database table has the following syntax:

CREATE TABLE *table-name*
(*column-definition-list*)

table name:

Oracle permits names of up to 30 characters long, which can consist of letters, numbers and underscores. Spaces are not allowed, and the first character should be a letter. Table names, like all user-defined names, should be meaningful. If you use abbreviations, then be consistent, and ensure that their meaning is obvious.

column-definition-list:

This consists, for each column of the table:

column name *data type* *constraint*

Column names follow the same rules as table names.
Oracle data types are shown in Appendix 2

***Example 3.6.1*: Let us consider the CREATE statement used to create the Airport table for the Airline database**

```
CREATE TABLE AIRPORT
  (AIRPORT_CODE     VARCHAR2(4)     PRIMARY KEY,
   AIRPORT_NAME     VARCHAR2(20)    UNIQUE,
   CHECK_IN         VARCHAR(50)     NOT NULL,
   RESERVATIONS     VARCHAR2(12),
   FLIGHT_INFO      VARCHAR2(12))
```

This SQL statement defines a table with 5 columns, of which the first (*Airport*), holds the primary key.

The AIRPORT_NAME column has a uniqueness constraint, which means that no two rows in the table may contain the same value for this column (i.e. no two airports may have the same name). This will be checked by the DBMS when data is entered, and duplicated values will cause an error message to be generated.

The CHECK_IN column has a NOT NULL constraint. This means that every row in the table must have a value for this column. Again, the DBMS will generate an error message if an attempt is made to enter a new row with no value for CHECK_IN (or if an attempt is made to delete the CHECK_IN value for an existing row).

3.6.2 Removing Unwanted Tables

When a database table becomes redundant it may be dropped from the database as follows:

DROP TABLE *table-name*

3.6.3 Tables with Composite Primary Keys

Example 3.6.3: Creating a table with a composite primary key

The following syntax is used:

```
CREATE TABLE TARIFF
        (ROUTE_NO   NUMBER(3),
        FARE_TYPE   CHAR(3),
        PRICE       NUMBER(4,2)   NOT NULL,
        CONSTRAINT  PK_TARIFF  PRIMARY  KEY  (ROUTE_NO,
        FARE_TYPE));
```

This statement not only creates a table and defines a primary key constraint on the columns ROUTE_NO and FARE_TYPE, but also gives it a name (PK_TARIFF). The constraint may then be referred to by name in any error messages generated by the DBMS.

3.7 Changing the Structure of Tables

It may become necessary from time to time to change the structure of tables within a database, to meet the needs of its users. This practice must be centrally controlled (by the database administrator), and should not be undertaken lightly, because:

Existing data may be affected
Programs that access the database may need to be modified

A well-designed database will minimize the need for such changes.

Adding Columns

The syntax for adding columns to a table is as follows:

ALTER TABLE *tablename* **ADD**
(column-definition-list)

Example 3.7.1: - The following statement adds two columns to the end of the Passenger table:

```
ALTER TABLE PASSENGER ADD
   (EMAIL VARCHAR2(25) NOT NULL,
   D_O_B DATE)
```

Removing Columns

Single columns can be removed as follows:

ALTER TABLE *tablename*
DROP COLUMN *columnname*

as in the following example:

Example 3.7.2(a): - To remove one of the columns added to the Passenger table in the previous example:

```
ALTER TABLE PASSENGER
DROP COLUMN EMAIL
```

Multiple columns can be removed using the following syntax. Note in particular that the keyword COLUMN is not used:

ALTER TABLE *tablename*
DROP (*column-list*)

where column names are separated by commas, as in this example:
Example 3.7.2(b): - To remove both of the columns added to the Passenger table in Example 3.7.1:

```
ALTER TABLE PASSENGER
DROP (EMAIL, D_O_B)
```

Modifying Columns

The following syntax is used to replace one or more column definitions with new ones:

ALTER TABLE *tablename* **MODIFY**
(*column-definition-list*)

The following example assumes that Price previously had the datatype and qualifier of NUMBER(4,2):

Example 3.7.3: - To change the column definition of Price in the Tariff table:

```
ALTER TABLE TARIFF MODIFY
(PRICE NUMBER(5,2))
```

Limitations

While the syntax for altering database tables is simple, it is necessary to consider the effect of making alterations on any existing data.

The DBMS will not allow the addition a column with a NOT NULL constraint (as in Example 3.7.1.) if the table already contains data. This is because it would result in rows with null values - thus violating the constraint. (Although it is possible to accomplish the addition in several stages).

Removing columns, as in Examples 3.7.2(a) and 3.7.2(b) may result in existing data being lost.

The DBMS will protect any existing data in a table column by preventing the following modifications from being made:

A change of data type
A decrease in the number of digits for numeric data
A decrease in the number of characters for textual data

EXERCISE 2

Write SQL statements to create all the tables of the Airline database, following examples 3.6.1 and 3.6.3.