ORACLE 10g/11g
LAB 3-4

Mr Ajit Gopee

Lecturer
School of Innovative Technologies and Engineering
University of Technology, Mauritius)
E-Mail: agopee@umail.utm.ac.mu

Contents

---

1. Introduction

The Oracle Relational Database Management System (RDBMS) is an industry leading database system designed for mission critical data storage and retrieval. The RDBMS is responsible for accurately storing data and efficiently retrieving that data in response to user queries.

The Oracle Corporation also supplies interface tools to access data stored in an Oracle database. Two of these tools are known as Oracle 10g, a command line interface within a Windows browser based interface , and Developer/2000 (now called simply *Developer*), a collection of forms, reports and graphics interfaces. This technical working paper introduces the features of the Oracle 10g and provides a tutorial that demonstrates its salient features.

This tutorial is intended for students and database practitioners who require an introduction to SQL, an introduction to working with the Oracle 10g.

This document is organized as follows. A brief overview of the suite of Oracle products is first presented in Section 2. In Section 3, we discuss the basics of working with the Oracle 10g . Structured Query Language (SQL), including data definition language (DDL) and data manipulation language (DML) is discussed in section 4. Advanced SQL commands are discussed in section 5 and a brief introduction to stored procedures and triggers is given in section 6.

---

2. Oracle Products: An Overview

The Oracle products suite includes the following tools and utilities:

2.1 Application Development Tools

- **Oracle 10g** - A command line tool used to manipulate tables and other database objects in an Oracle database.
- **Developer/2000 and Developer** A suite of application development tools including Forms, Reports and Graphics.
    - **Oracle*Forms** - A screen based tool used to develop data entry forms and menus that access tables in an Oracle database.
    - **Oracle*Reports** - A screen based tool used to develop reports that access tables in an Oracle database.
    - **Oracle*Graphics** - A graphical tool used to develop charts and reports that access tables in an Oracle database.
- **CASE*Designer and Oracle Designer/2000** - A graphical tool used to create and display models contained in the CASE*Dictionary.
- **CASE*Dictionary** - A repository for business rules, functional models and data models used for organizing and documenting an application development effort.
- **CASE*Generator** - A code generating tool that uses information stored in CASE*Dictionary to develop data entry forms, reports and graphics.
- **Oracle*Book** - A graphical tool used to develop on-line documentation with hypertext capabilities.
- **SQL*TextRetrieval and Oracle Context** - A suite of tools and API used to develop sophisticated text search and retrieval applications.
- **Programmer/2000** - Including the Pro* precompilers - Libraries of routines and utilities that can be linked with ``C", C++, FORTRAN, Java, ADA, COBOL or other host languages to allow access to Oracle databases.

2.2 Database Utilities

- **Enterprise Manager** - A GUI based collection of utilities for managing Oracle Databases.
- **SQL*DBA and SVRMGR** - A utility that allows the database administrator (DBA) to monitor database activity and to tune the database for optimal performance.
- **Export/Import** - Command line utilities that allow a user or the DBA to export data from an Oracle database into a machine readable file or to import data from a machine readable file into an Oracle database.
- **SQL*Loader -** A command line utility to load ASCII or binary data files into an Oracle database.
- **Oracle*Terminal** - A utility program used to customize the user interface and keyboard mappings for all Oracle tools. This utility allows all Oracle tools to have a similar ``look and feel" across many different hardware and operating system platforms.

2.3 Connectivity and Middleware Products

- **SQL*Net and Net8** - A communications driver that allows an Oracle tool running on a client machine to access Oracle data on a separate server machine.
- **SQL*Connect and Oracle Gateways** - A communications driver that allows an Oracle tool running on a client machine to access Non-Oracle data on a

server machine such as data residing in a DB2 database or MS SQL Server database.

- **ORACLE Server** - Typically a part of the Oracle RDBMS running on a database server, this component receives requests from client machines and submits them to the Oracle RDBMS. The results are then passed back to the client machines.
- **Oracle ODBC Drivers** - Open DataBase Connectivity drivers for connecting software to Oracle databases using the ODBC standard.

2.4 Core Database Engine

- **ORACLE RDBMS** - The Oracle Relational Database Engine. Now called the Oracle Universal Server with several options in addition to managing relational data. These options are now called *Cartridges*:
    - **Oracle Web Applications Server** - A WWW Server (HTTP server) linked into the Oracle RDBMS. Allows web based applications using HTML forms and JAVA to access and manipulate data.
    - **Spatial Data Cartridge**- Allows storage of temporal and spatial data in the Oracle RDBMS. Useful for Geographic Information Systems (GIS).
    - **Video Cartridge** - Provides storage and real-time serving of streaming video.
    - **ConText Cartridge** - Provides storage and retrieval of text documents.
    - **Messaging Option** - A groupware architecture built on top of the RDBMS.
    - **OLAP Option** - Tools and database support for On-Line Analytical Processing.
    - **Objects Option** - Allows complex objects to be modeled and stored in the DBMS. Includes Object Oriented features such as encapsulation, inheritance, server and client side methods, etc.
- **Integrated Data Dictionary** - Stores and manages access to all of the tables owned by all users in a system.
- **SQL** - The language used to access and manipulate database data.
- **PL/SQL** - A procedural extension to the SQL language unique to the Oracle line of products.

2.5 Typical Development Environments

Developing applications using an Oracle database requires access to a copy of the Oracle RDBMS (or a central Oracle RDBMS server), and one or more of the development tools. Third party development tools such as PowerBuilder, Visual Basic or Java can also be used for applications development.

Stand-alone development in a single user environment can be accomplished using the Personal Oracle or Personal Oracle Lite RDBMS in conjunction with Oralce Developer or a third party development tool.

Muli-user development in a shared environment can be accomplished using an Oracle RDBMS server running on a server machine. Distributed client PCs can develop the applications using any of the tools mentioned above.

Regardless of the development environment, used, the Oracle 10g is a convenient and capable tool for manipulating data in an Oracle database. In the following section, the Oracle 10g is introduced.

3. ORACLE 10g Basics

Oracle 10g is a command line tool that allows a user to type SQL statements to be executed directly against an Oracle database. Oracle 10g has the ability to format database output, save often used commands and can be invoked from other Oracle tools or from the operating system prompt.

In the following sections, the basic functionality of Oracle 10g will be demonstrated along with sample input and output to demonstrate some of the many features of this product.

3.1 Running Oracle 10g

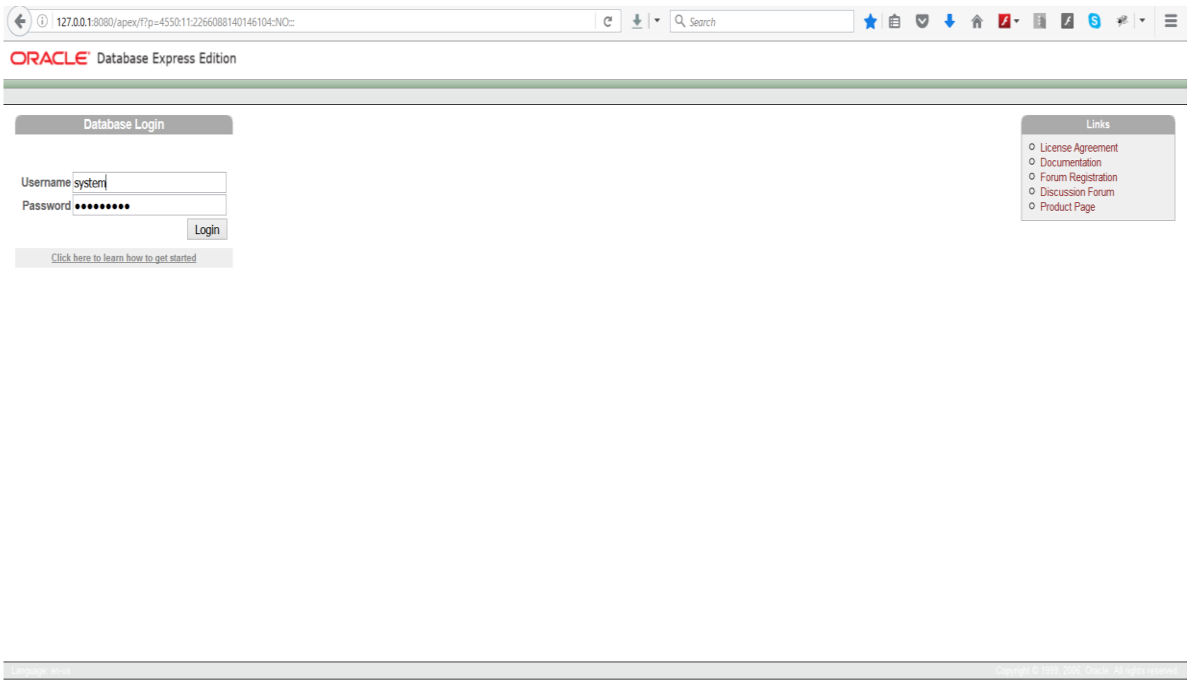
In this section, we give some general directions on how to get into the Oracle 10g program and connect to an Oracle database..

Before using oracle 10g or any other development tool or utility, the user must obtain an Oracle account for the DBMS. This account will include a username, a password and, optionally, a host string indicating the database to connect to. This information can typically be obtained from the database administrator.

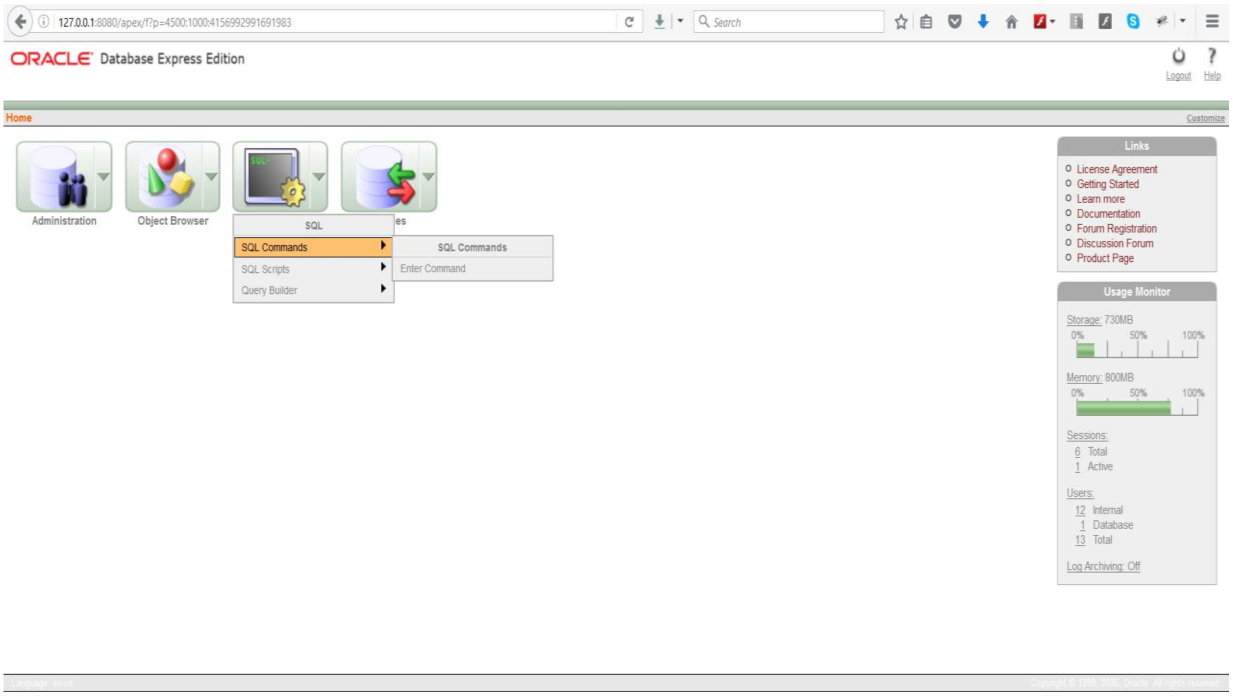
3.1.1 Running Oracle 10g under Windows.

1. Click on start
2. Click on Program
3. Click on Oracle Database 10g Express Edition
4. Click Go to Database Home Page

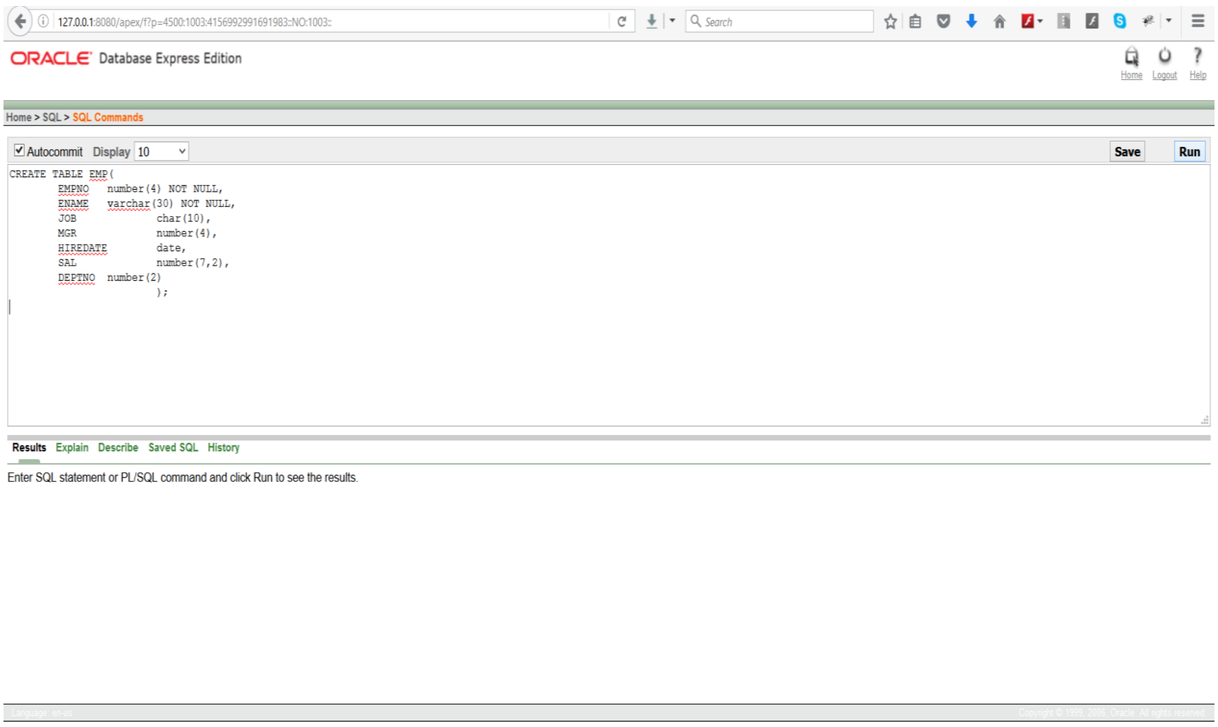
   This will take you to the following:



5. Enter your username and password



6. Click on Login
7. Click on SQL --→ SQL Commands -→ Enter the following SQL Commands and click on the button Run


127.0.0.1:8080/apex/f?p=4500:1003:4156992991691983::NO:1003::
Database Express Edition
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
CREATE TABLE EMP(
        EMPNO   number(4) NOT NULL,
        ENAME   varchar(30) NOT NULL,
        JOB             char(10),
        MGR             number(4),
        HIREDATE        date,
        SAL             number(7,2),
        DEPTNO  number(2)
                        );
```

Enter SQL statement or PL/SQL command and click Run to see the results.

To exit the SQL program (in any operating system), type `EXIT` and press Enter or carriage return:

```
SQL>   EXIT
```

Once a session has been established using the SQL*Plus tool, any SQL statements or SQL*Plus Commands may be issued. In the following section, the basic SQL*Plus Commands are introduced.

3.2 SQL Commands

SQL commands allow a user to manipulate and submit SQL statements. Specifically, they enable a user to:

- Enter, edit, store, retrieve, and run SQL statements
- List the column definitions for any table
- Format, perform calculations on, store, and print query results in the form of reports
- Access and copy data between SQL databases

The following is a list of SQL commands and their functions. The most commonly used commands are emphasized in italics:

- / - Execute the current SQL statement in the buffer - same as RUN
- ACCEPT - Accept a value from the user and place it into a variable
- APPEND - Add text to the end of the current line of the SQL statement in the buffer

- AUTOTRACE - Trace the execution plan of the SQL statement and gather statistics
- BREAK - Set the formatting behavior for the output of SQL statements
- BTITLE - Place a title on the bottom of each page in the printout from a SQL statement
- CHANGE - Replace text on the current line of the SQL statement with new text
- CLEAR - Clear the buffer
- COLUMN - Change the appearance of an output column from a query
- COMPUTE - Does calculations on rows returned from a SQL statement
- CONNECT - Connect to another Oracle database or to the same Oracle database under a different user name
- COPY - Copy data from one table to another in the same or different databases
- DEL - Delete the current line in the buffer
- DESCRIBE - List the columns with datatypes of a table
- EDIT - Edit the current SQL statement in the buffer using an external editor such as `vi` or `emacs`
- EXIT - Exit the SQL*Plus program
- GET - Load a SQL statement into the buffer but do not execute it
- *HELP* - Obtain help for a SQL*Plus command (In some installations)
- HOST - Drop to the operating system shell
- INPUT - Add one or more lines to the SQL statement in the buffer
- LIST - List the current SQL statement in the buffer
- QUIT - Exit the SQL*Plus program
- REMARK - Place a comment following the REMARK keyword
- *RUN* - Execute the current SQL statement in the buffer
- *SAVE* - Save the current SQL statement to a script file
- SET - Set a variable to a new value
- SHOW - Show the current value of a variable
- *SPOOL* - Send the output from a SQL statement to a file
- *START* - Load a SQL statement located in a script file and then run that SQL statement
- TIMING - Used to time the execution of SQL statements for performance analysis
- TTITLE - Place a title on the top of each page in the printout from a SQL statement
- UNDEFINE - Delete a user defined variable

Examples of these SQL*Plus commands are given in the following sections.

---

4. The SQL Language

Structured Query Language (SQL) is the language used to manipulate relational databases. SQL is tied very closely with the relational model.

In the relational model, data is stored in structures called relations or *tables*. Each table has one or more attributes or *columns* that describe the table. In relational databases, the table is the fundamental building block of a database application. Tables are used to store data on Employees, Equipment, Materials, Warehouses, Purchase Orders, Customer Orders, etc. Columns in the Employee table, for example, might be Last Name, First Name, Salary, Hire Date, Social Security Number, etc.

SQL statements are issued for the purpose of:

- Data definition - Defining tables and structures in the database (DB).
- Data manipulation - Inserting new data, Updating existing data, Deleting existing data, and Querying the Database (Retrieving existing data from the database).

Another way to say this is the SQL language is actually made up of 1) the Data Definition Language (DDL) used to create, alter and drop scema objects such as tables and indexes, and 2) The Data Manipulation Language (DML) used to manipulate the data within those schema objects.

The SQL language has been standardized by the ANSI X3H2 Database Standards Committee. Two of the latest standards are SQL-89 and SQL-92. Over the years, each vendor of relational databases has introduced new commands to extend their particular implementation of SQL. Oracle's implementation of the SQL language conforms to the basic SQL-92 standard and adds some additional commands and capabilities.

4.1 SQL Statements

The following is an alphabetical list of SQL statements that can be issued against an Oracle database. These commands are available to any user of the Oracle database. Emphasized items are most commonly used.

- ALTER - Change an existing table, view or index definition
- AUDIT - Track the changes made to a table
- COMMENT - Add a comment to a table or column in a table
- COMMIT - Make all recent changes permanent
- *CREATE* - Create new database objects such as tables or views
- *DELETE* - Delete rows from a database table
- *DROP* - Drop a database object such as a table, view or index
- GRANT - Allow another user to access database objects such as tables or views
- *INSERT* - Insert new data into a database table
- No AUDIT - Turn off the auditing function
- REVOKE - Disallow a user access to database objects such as tables and views
- ROLLBACK - Undo any recent changes to the database
- *SELECT* - Retrieve data from a database table
- *UPDATE* - Change the values of some data items in a database table

Some examples of SQL statements follow. For all examples in this tutorial, key words used by SQL and Oracle are given in all uppercase while user-specific information, such as table and column names, is given in lower case.

To create a new table to hold employee data, we use the CREATE TABLE statement:

```
   CREATE TABLE employee
   (fname           VARCHAR2(8),
    minit           VARCHAR2(2),
    lname           VARCHAR2(8),
    ssn             VARCHAR2(9) NOT NULL,
    bdate           DATE,
    address         VARCHAR2(27),
    sex             VARCHAR2(1),
    salary          NUMBER(7) NOT NULL,
    superssn        VARCHAR2(9),
    dno             NUMBER(1) NOT NULL) ;
```

To insert new data into the employee table, we use the INSERT statement:

```
INSERT INTO employee
VALUES ('BUD', 'T', 'WILLIAMS', '132451122',
       '24-JAN-54', '987 Western Way, Plano, TX',
       'M', 42000, NULL, 5);
```

To retrieve a list of all employees with salary greater than 30000 from the employees table, the following SQL statement might be issued (Note that all SQL statements end with a semicolon):

```
          SELECT fname, lname, salary
          FROM   employee
          WHERE  salary > 30000;
```

To give each employee in department 5 a 4 percent raise, the following SQL statement might be issued:

```
          UPDATE employee
          SET    salary = salary * 1.04
          WHERE  dno = 5;
```

To delete an employee record from the database, the following SQL statement might be issued:

```
          DELETE FROM employee
          WHERE  empid = 101 ;
```

The above statements are just an example of some of the many SQL statements and variations that are used with relational database management systems. The full syntax of these commands and additional examples are given below.

4.2 SQL Data Definition Language

In this section, the basic SQL Data Definition Language statements are introduced and their syntax is given with examples.

An Oracle database can contain one or more *schemas*. A schema is a collection of database objects that can include: tables, views, indexes and sequences. By default, each user has their own the schema which has the same name as the Oracle username. For example, a single Oracle database can have separate schemas for HOLOWCZAK, JONES, SMITH and GREEN.

Any object in the database must be created in only one schema. The object name is prefixed by the schema name as in: `schema.object_name`
By default, all objects are created in the user's own schema. For example, when JONES creates a database object such as a table, it is created in her own schema. If JONES creates an EMPLOYEE table, the full name of the table becomes: `JONES.EMPLOYEE`. Thus database objects with the same name can be created in more than one schema. This feature allows each user to have their own EMPLOYEE table, for example.

Database objects can be shared among several users by specifying the schema name. In order to work with a database object from another schema, a user must be granted authorization. See the section below on GRANT and REVOKE for more details.

Please note that many of these database objects and options are not available under Personal Oracle Lite. For example, foreign key constraints are not supported. Please see the on-line documentation for Personal Oracle Lite for more details.

4.2.1 Create, Modify and Drop Tables, Views and Sequences

SQL*Plus accepts SQL statements that allow a user to create, alter and drop table, view and sequence definitions. These statements are all standard ANSI SQL statements with the exception of CREATE SEQUENCE.

- **ALTER TABLE** - Change an existing table definition. The table indicated in the ALTER statement must already exist. This statement can be used to add a new column or remove an existing column in a table, modify the data type for an existing column, or add or remove a constraint.

  ALTER TABLE has the following syntax for adding a new column to an existing table:

  ```
  ALTER TABLE
  ```
  ADD ( ) ;
  Another ALTER TABLE option can change a data type of column. The syntax is:
  ```
  ALTER TABLE
  ```
  MODIFY ( ) ;

Finally, ALTER TABLE can also be used to add a constraint to a table such as for a PRIMARY KEY, FOREIGN KEY or CHECK CONSTRAINT. The syntax to add a PRIMARY KEY is:

```
          ALTER TABLE
```

ADD CONSTRAINT PRIMARY KEY ();
The syntax to add a FOREIGN KEY constraint is:

```
          ALTER TABLE
          ADD CONSTRAINT
          FOREIGN KEY ()
          REFERENCES  (column-name);
```

In Oracle, you must use an ALTER TABLE statement to define a composite PRIMARY KEY (a key made up of two or more columns).

NOTE: In Oracle, there is no single command to drop a column of a table. In order to drop a column from a table, you must create a temporary table containing all of the columns and records that will be retained. Then drop the original table and rename the temporary table to the original name. This is demonstrated below in the section on Creating, Altering and Dropping Tables.

- **CREATE TABLE** - Create a new table in the database. The table name must not already exist. CREATE TABLE has the following syntax:

```
          CREATE TABLE
          (     ,
                ,
            . . .
          ) ;
```

An alternate syntax can be used to create a table with a subset of rows or columns from an existing table.

```
          CREATE TABLE  AS
           ;
```

- **DROP TABLE** - Drop a table from the database. The table name must already exist in the database. The syntax for the DROP TABLE statement is:

```
          DROP TABLE
```

;

- **CREATE INDEX** - Create a new Index that facilitates rapid lookup of data. An index is typically created on the primary and/or secondary keys of the table. The basic syntax for the CREATE INDEX statement is:

```
          CREATE INDEX
          ON
```

(,) ;

- **DROP INDEX** - Drop an index from the database. The syntax for the DROP INDEX statement is:

```
          DROP INDEX  ;
```

- **CREATE SEQUENCE** - Create a new Oracle Sequence of values. The new sequence name must not exist. CREATE SEQUENCE has the following syntax:

```
CREATE SEQUENCE
INCREMENT BY
START WITH
MAXVALUE
CYCLE ;
```

- **DROP SEQUENCE** - Drop an Oracle Sequence. The sequence name must exist. DROP SEQUENCE has the following syntax:

```
DROP SEQUENCE  ;
```

- **CREATE VIEW** - Create a new view based on existing tables in the database. The table names must already exist. The new view name must not exist. CREATE VIEW has the following syntax:

```
CREATE VIEW  AS
 ;
```

where *sql select statement* is in the form:

```
SELECT
FROM
```

WHERE

Additional information on the SELECT statement and SQL queries can be found in the next section.

Note that an ORDER BY clause may not be added to the *sql select statement* when defining a view.

In general, views are read-only. That is, one may query a view but it is normally the case that views can not be operated on with INSERT, UPDATE or DELETE. This is especially true in cases where views joing two or more tables together or when a view contains an aggregate function.

- **DROP VIEW** - Drop a view from the database. The view name must already exist in the database. The syntax for the DROP VIEW command is:

```
DROP VIEW  ;
```

In the following section, each of the SQL DDL commands will be discussed in more detail.

Creating, Altering and Dropping Tables

A table is made up of one or more columns (also called attributes in relational theory). Each column is given a name and a data type that reflects the kind of data it will store. Oracle supports four basic data types called CHAR, NUMBER, DATE and RAW. There are also a few additional variations on the RAW and CHAR data types. The basic datatypes, uses and syntax, are as follows:

- **VARCHAR2** - Character data type. Can contain letters, numbers and punctuation. The syntax for this data type is: `VARCHAR2(size)` where *size* is the maximum number of alphanumeric characters the column can hold. For example `VARCHAR2(25)` can hold up to 25 alphanumeric characters. In Oracle8, the maximum size of a VARCHAR2 column is 4,000 bytes.

  The VARCHAR data type is a synonym for VARCHAR2. It is recommended to use VARCHAR2 instead of VARCHAR.

- **NUMBER** - Numeric data type. Can contain integer or floating point numbers only. The syntax for this data type is: `NUMBER(precision, scale)` where *precision* is the total size of the number including decimal point and *scale* is the number of places to the right of the decimal. For example, `NUMBER(6,2)` can hold a number between `-999.99` and `999.99`.
- **DATE** - Date and Time data type. Can contain a date and time portion in the format: `DD-MON-YY HH:MI:SS`. No additional information is needed when specifying the DATE data type. If no time component is supplied when the date is inserted, the time of 00:00:00 is used as a default. The output format of the date and time can be modified to conform to local standards.
- **RAW** - Free form binary data. Can contain binary data up to 255 characters. Data type LONG RAW can contain up to 2 gigabytes of binary data. RAW and LONG RAW data cannot be indexed and can not be displayed or queried in SQL*Plus. Only one RAW column is allowed per table.
- **LOB** - Large Object data types. These include BLOB (Binary Large OBject) and CLOB (Character Large OBject). More than one LOB column can appear in a table. These data types are the prefferred method for storing large objects such as text documents (CLOB), images, or video (BLOB).

A column may be specified as NULL or NOT NULL meaning the column may or may not be left blank, respectively. This check is made just before a new row is inserted into the table. By default, a column is created as NULL if no option is given.

In addition to specifying NOT NULL constraints, tables can also be created with constraints that enforce referential integrity (relationships among data between tables). Constraints can be added to one or more columns, or to the entire table.

Each table may have one PRIMARY KEY that consists of a single column containing no NULL values and no repeated values. A

PRIMARY KEY with multiple columns can be identified using the ALTER TABLE command.

Up to 255 columns may be specified per table. Column names and table names must start with a letter and may not contain spaces or other punctuation except for the underscore character. Column names and table names are case insensitive. This means that you can specify the names of columns and tables in any way you like. For example, the following three SELECT statements are all identical:

```
SELECT lname, fname, address FROM employee;
SELECT LNAME, FNAME, ADDRESS FROM EMPLOYEE;
SELECT Lname, Fname, Address FROM Employee;
```

In the following example, a new table called ``employee" is created with ten columns of a variety of types. The columns indicated by NOT NULL will be mandatory while the other columns, by default, will be optional.

```
SQL>   CREATE TABLE employee
   2   (fname           VARCHAR2(8),
   3    minit           VARCHAR2(2),
   4    lname           VARCHAR2(8),
   5    ssn             VARCHAR2(9) NOT NULL,
   6    bdate           DATE,
   7    address         VARCHAR2(27),
   8    sex             VARCHAR2(1),
   9    salary          NUMBER(7) NOT NULL,
  10    superssn        VARCHAR2(9),
  11    dno             NUMBER(1) NOT NULL) ;

 Table created.

 SQL>
```

The numbers 2 through 11 before each line indicate the line number supplied by the SQL*Plus program as this statement was typed in. We will omit these numbers in the rest of the examples to facilitate copying and pasting this material directly into a live SQL*Plus session.

A new table can also be created with a subset of the columns in an existing table. In the following example, a new table called `emp_department_1` is created with only the *fname, minit, lname* and *bdate* columns from the `employee` table. This new table is also populated with data from the `employee` table where the employees are from department number 1.

```
SQL> CREATE TABLE emp_department_1
     AS SELECT fname, minit, lname, bdate
     FROM employee
     WHERE dno = 1 ;

 Table created.

SQL> DESCRIBE emp_department_1
 Name                            Null?    Type
 ------------------------------- -------- ----
 FNAME                                    VARCHAR2(8)
 MINIT                                    VARCHAR2(2)
 LNAME                                    VARCHAR2(8)
 BDATE                                    DATE

SQL>
```

One can also create a new table with all of the columns from the original table, but with only a subset of the rows form the original table:

```
SQL> CREATE TABLE high_pay_emp
     AS SELECT *
     FROM employee
     WHERE salary > 50000 ;

 Table created.
```

DESCRIBE is an SQL*Plus command that displays the columns of a table and their data types. The syntax for the DESCRIBE command is:

```
   DESCRIBE
;
```

The copying of data can be suppressed by giving a WHERE clause that always evaluates to FALSE for each record in the source table. The following example makes a duplicate of the `employee` table but does not copy any data into it.

```
SQL>  CREATE TABLE copy_of_employee
      AS SELECT *
      FROM employee
      WHERE 3=5 ;

 Table created.

SQL> DESCRIBE copy_of_employee
 Name                            Null?    Type
 ------------------------------- -------- ----
 FNAME                                    VARCHAR2(8)
 MINIT                                    VARCHAR2(2)
 LNAME                                    VARCHAR2(8)
 SSN                             NOT NULL VARCHAR2(9)
 BDATE                                    DATE
 ADDRESS                                  VARCHAR2(27)
 SEX                                      VARCHAR2(1)
 SALARY                          NOT NULL NUMBER(7)
 SUPERSSN                                 VARCHAR2(9)
 DNO                                NOT NULL NUMBER(1)
```

Constraints can be added to the table at the time it is created, or at a later time using the ALTER TABLE statement. Constraints can include:

- Primary key and Unique key constraints.
- Foreign key constraints (for referential integrity).
- Check constraints.

Here is an example of creating a primary key constraint on the `empid` column:

```
CREATE TABLE employee
(fname           VARCHAR2(8),
 minit           VARCHAR2(2),
 lname           VARCHAR2(8),
 ssn             VARCHAR2(9) NOT NULL,
 bdate           DATE,
 address         VARCHAR2(27),
 sex             VARCHAR2(1),
 salary          NUMBER(7) NOT NULL,
 superssn        VARCHAR2(9),
 dno             NUMBER(1) NOT NULL,
 CONSTRAINT pk_emp PRIMARY KEY (ssn) );
```

Referential integrity constraints can also be added. In the following example, the `dno` column in the `employee` table references the `dnumber` column in the `department` table. If a department is deleted, all employees that reference the department are also deleted. This is given by the `ON DELETE CASCADE` option:

```
CREATE TABLE department
(dnumber      NUMBER(1),
 dname        VARCHAR2(15),
 mgrssn       VARCHAR2(9),
 mgrstartdate DATE
 CONSTRAINT pk_department PRIMARY KEY (dnumber) );

CREATE TABLE employee
(fname           VARCHAR2(8),
 minit           VARCHAR2(2),
 lname           VARCHAR2(8),
 ssn             VARCHAR2(9) NOT NULL,
 bdate           DATE,
 address         VARCHAR2(27),
 sex             VARCHAR2(1),
 salary          NUMBER(7) NOT NULL,
 superssn        VARCHAR2(9),
 dno             NUMBER(1) NOT NULL,
 CONSTRAINT pk_emp PRIMARY KEY (ssn),
 CONSTRAINT fk_dno FOREIGN KEY (dno)
 REFERENCES department (dnumber) ON DELETE CASCADE);
```

In order to specify a foreign key constraint, the column in the child (or detail) table (e.g., the `dnumber` column in the `department` table in the above example) must be either the primary key or a unique key for the table. Thus, the child (or detail) table must be created first before the parent (or master) table is created using the above constraints.

Additional CREATE TABLE constraint statements allow the specification of what should happen when a row is deleted or updated in a parent table. In the above example, deleting a department causes all employees in that department to also be deleted. Other options include `ON DELETE SET DEFAULT` and `ON DELETE SET NULL`. In addition, the behavior of child tables when a parent table is updated can also be specified using an `ON UPDATE` clause.

CHECK constraints can be added to check the values for a given column. This can be used to allow only a specific set of valid values for a column. In the following example, CHECK constraints are added to limit the valid values for the `sex` column and to check if the salary is greater than 10,000 (be sure to DROP TABLE employee before you try the next one).

```
CREATE TABLE employee
(fname          VARCHAR2(8),
 minit          VARCHAR2(2),
 lname          VARCHAR2(8),
 ssn            VARCHAR2(9) NOT NULL,
 bdate          DATE,
 address        VARCHAR2(27),
 sex            VARCHAR2(1)
     CONSTRAINT ck_sex CHECK (sex IN ('M', 'F')),
 salary         NUMBER(7) NOT NULL
     CONSTRAINT ck_salary CHECK (salary > 10000),
 superssn       VARCHAR2(9),
 dno            NUMBER(1) NOT NULL,
 CONSTRAINT pk_emp PRIMARY KEY (ssn),
 CONSTRAINT fk_dno FOREIGN KEY (dno)
    REFERENCES department (dnumber) ON DELETE CASCADE);
```

The CHECK constraints are activated when inserting a new row or when updating existing data. In the following example, the value given for sex is 'm':

```
SQL> insert into employee values
  2  ('Joe', 'M', 'Smith', '123456789', '01-JUN-45',
  3   '123 Smith St.', 'm', 45000, '123456789', 1) ;
  insert into employee values
  *
  ERROR at line 1:
  ORA-02290: check constraint (HOLOWCZAK.CK_SEX) violated
```

In the previous examples, constraints were given names with the following prefixes:

- Primary key constraints: `pk_`
- Foreign key constraints: `fk_`
- Check constraints: `ck_`

Naming constraints in this fashion is simply a convenience. Any name may be given to a constraint.

The ALTER TABLE command can be used to add a new column to an existing table or to change the data type of an existing column. The following examples add a new column *manager* to an existing table named `emp_department_1` and then modify the data type of the *fname* column.

```
SQL> DESCRIBE emp_department_1
 Name                            Null?    Type
 ------------------------------- -------- ----
 FNAME                                    VARCHAR2(8)
 MINIT                                    VARCHAR2(2)
 LNAME                                    VARCHAR2(8)
 BDATE                                    DATE

SQL>  ALTER TABLE emp_department_1
      ADD (manager VARCHAR2(8)) ;

Table altered.

SQL>  ALTER TABLE emp_department_1
      MODIFY (fname VARCHAR2(15));

Table altered.

SQL> DESCRIBE emp_department_1
 Name                            Null?    Type
 ------------------------------- -------- ----
 FNAME                                    VARCHAR2(15)
 MINIT                                    VARCHAR2(2)
 LNAME                                    VARCHAR2(8)
 BDATE                                    DATE
 MANAGER                                  VARCHAR2(8)
```

The ALTER TABLE command can also be used to change the datatype of column *provided there is no data in the table*. To get around this if there is data in the table, create a temporary table using all of the data from the existing table, delete the existing records from the original table, alter the datatype, and then insert the records from the temporary table back into the original table. For example, assume the `emp_department_1` table has some records in it and we want to change the datatype for the MANAGER column:

```
CREATE TABLE temp AS SELECT * FROM emp_department_1;

DELETE FROM emp_department_1;

ALTER TABLE emp_department_1
MODIFY (manager VARCHAR2(15));

INSERT INTO emp_department_1
SELECT * FROM temp;

DROP TABLE temp;
```

This trick can also be used to drop a column from a table. Assume the Employee table has the following columns: `fname, minit, lname, ssn, bdate, address, sex, salary, superssn` and `dno`, and we want to drop the `salary` column from the table:

```
CREATE TABLE temp AS
SELECT fname, minit, lname, ssn, bdate,
address, sex, superssn, dno FROM employee;

DROP TABLE employee;

CREATE TABLE employee AS
SELECT * FROM temp;
```

The DROP TABLE command can be used to drop a table definition and all of its data from the database. In the following example, the table `emp_department_1` created previously, is dropped from the database.

```
SQL> DROP TABLE emp_department_1 ;

Table dropped.
```

Exercise 1: Creating and Altering Tables

As an exercise, create a table called STUDENTS with the following columns and data types:

```
Column Name      Data Type
StudentID        NUMBER(5,0)       NOT NULL
Name             VARCHAR2(25)
Major            VARCHAR2(15)
GPA              NUMBER(6,3)
```

Create another table called COURSES with the following columns and data types:

```
Column Name      Data Type
```

```
StudentID        NUMBER(5,0)        NOT NULL
CourseNumber     VARCHAR2(15)       NOT NULL
CourseName       VARCHAR2(25)
Semester         VARCHAR2(10)
Year             NUMBER(4,0)
Grade            VARCHAR2(2)
```

Use the DESCRIBE command to display the data types of the columns after each table is created.

Next, use the ALTER TABLE statement to add the following column to the STUDENTS table:

```
Column Name      Data Type
TutorID          NUMBER(5,0)
```

Use the ALTER TABLE statement to define the StudentID as the PRIMARY KEY for the STUDENTS table.

Use the ALTER TABLE statement to define the StudentID and CourseNumber as the PRIMARY KEY for the COURSES table. To do this, list both of the column names separated by a comma.

Use the ALTER TABLE statement to define StudentID in the COURSES table as a FOREIGN KEY that references the StudentID in the STUDENTS table.

Finally, add some data to the STUDENTS and COURSES tables (simply copy and paste these statements into SQL*Plus to add the data):

```
INSERT INTO students VALUES (101, 'Bill', 'CIS', 3.45,
102);
INSERT INTO students VALUES (102, 'Mary', 'CIS', 3.10,
NULL);
INSERT INTO students VALUES (103, 'Sue',  'Marketing',
2.95, 102);
INSERT INTO students VALUES (104, 'Tom',  'Finance', 3.5,
106);
INSERT INTO students VALUES (105, 'Alex', 'CIS', 2.75,
106);
INSERT INTO students VALUES (106, 'Sam',  'Marketing',
3.25, 103);
INSERT INTO students VALUES (107, 'Jane', 'Finance', 2.90,
102);

INSERT INTO courses VALUES (101, 'CIS3400', 'DBMS I',
'FALL', 1997, 'B+');
INSERT INTO courses VALUES (101, 'CIS3100', 'OOP I',
'SPRING', 1999, 'A-');
INSERT INTO courses VALUES (101, 'MKT3000', 'Marketing',
'FALL', 1997, 'A');
INSERT INTO courses VALUES (102, 'CIS3400', 'DBMS I',
'SPRING', 1997, 'A-');
INSERT INTO courses VALUES (102, 'CIS3500', 'Network I',
'SUMMER', 1997, 'B');
INSERT INTO courses VALUES (102, 'CIS4500', 'Network II',
'FALL', 1997, 'B+');
INSERT INTO courses VALUES (103, 'MKT3100', 'Advertizing',
'SPRING', 1998, 'A');
INSERT INTO courses VALUES (103, 'MKT3000', 'Marketing',
'FALL', 1997, 'A');
INSERT INTO courses VALUES (103, 'MKT4100', 'Marketing
II', 'SUMMER', 1998, 'A-');
```

Creating and Dropping Indexes

An index is a data structure that afford rapid lookup of data in a table. An index is normally created on those columns of a table used to look up data. For example, in the employee table, the key *ssn* can be used to look up the rest of an employee's information. Creating a index on the *ssn* field would be accomplished by the following statement:

```
SQL>  CREATE INDEX employee_ssn_idx
      ON employee (ssn) ;

Index created.
```

It is also possible to create indexes on other columns of a table. For example, if the `employee` table is frequently accessed by *superssn*, an index can be created on that column as well:

```
SQL>  CREATE INDEX employee_superssn_idx
      ON employee (superssn) ;

Index created.
```

Indexes can be dropped using the DROP INDEX statement: For example, to drop just the *employee_superssn_idx* index, one could submit:

```
DROP INDEX employee_superssn_idx ;

Index Dropped.
```

Note that dropping a table (using the DROP TABLE statement) automatically drops all indexes on that table.

Exercise 2: Creating and Altering Tables

For this exercise, create an index on the STUDENTS table for the `Name` column. Be sure to give this index an appropriate name.

Create an index on the COURSES table for the `semester` and `year` columns (together).

Creating and Dropping Views

In the SQL language, a view is a representation of one or more tables. A view can be used to hide the complexity of relationships between tables or to provide security for sensitive data in tables. In the following example, a limited view of the `employee` table is created. When a view is defined, a SQL statement is associated with the view name. Whenever the view is accessed, the SQL statement will be executed.

In the following example, the view `emp_dno_1` is created as a limited number of columns (*fname*, *lname*, *dno*) and limited set of data (`WHERE dno=1`) from the `employee` table.

```
     CREATE VIEW emp_dno_1
     AS SELECT fname, lname, dno
     FROM employee
     WHERE dno = 1;

View created.
```

Once the view is created, it can be queried with a SELECT statement as if it were a table.

```
SELECT * FROM emp_dno_1 ;

FNAME    LNAME          DNO
-------- -------- ---------
JAMES    BORG             1
```

Views can be dropped in a similar fashion to tables. The DROP VIEW command provides this facility. In the following example, the view just created is dropped.

```
DROP VIEW emp_dno_1 ;

View dropped.
```

Views can also be created to join several tables together. The following is an example of creating a view that joins two tables:

```
SQL> CREATE VIEW dept_managers AS
  2  SELECT dnumber, dname, mgrssn, lname, fname
  3  FROM   employee, department
  4  WHERE  employee.ssn = department.mgrssn ;

View created.

SQL> SELECT * FROM dept_managers ;

DNUMBER DNAME               MGRSSN LNAME    FNAME
------- --------------- ---------- -------- --------
      5 RESEARCH         333445555 WONG     FRANKLIN
      4 ADMINISTRATION   987654321 WALLACE  JENNIFER
      1 HEADQUARTERS     888665555 BORG     JAMES
```

This view can then be used as part of other queries or as the basis for developing applications.

As a final example, a view can be created that contains an aggregate function. In the following example, a view is created that returns the average salary of all employees per department.

```
SQL> CREATE VIEW dept_average_salary AS
  2  SELECT dnumber, dname, AVG(salary) AS average_salary
  3  FROM   department, employee
  4  WHERE  employee.dno = department.dnumber
  5  GROUP BY dnumber, dname ;

 View created.

SQL> SELECT * FROM dept_average_salary ;

   DNUMBER DNAME           AVERAGE_SALARY
---------- --------------- --------------
         1 HEADQUARTERS             55000
         4 ADMINISTRATION           31000
         5 RESEARCH                 33250
```

Note the use of the column alias `AS average_salary` and the mandatory GROUP BY clause.

Note that in general, views are read-only as in the above cases.

To see which views are defined in a schema, submit a query to the USER_VIEWS view:

```
SQL> SELECT view_name FROM user_views ;

VIEW_NAME
--------------------------
DEPT_AVERAGE_SALARY
DEPT_MANAGERS
EMP_DNO_1
```

Exercise 3: Creating Views

For this exercise, create a view called V_CIS_MAJORS basd upon the following SQL SELECT statement:

```
SELECT * FROM students WHERE major = 'CIS';
```

Query the view and show the output.

Create another view called V_COURSES_TAKEN based upon the following SQL SELECT statement:

```
SELECT name, major, coursenumber, coursename,
       semester, year, grade
FROM   students, courses
```

```
WHERE  students.studentid = courses.studentid;
```

Before querying this view, format the output column by submitting the following SQL*Plus COLUMN FORMAT commands:

```
COLUMN name       FORMAT A8
COLUMN coursename FORMAT A15
COLUMN major      FORMAT A10
COLUMN year       FORMAT 9999
```

As discussed in Section 5.2, the format command changes the way data is displayed in SQL*Plus. It does not change how the data is stored in the tables.

Query the V_COURSES_TAKEN view and show the output.

Creating, Altering and Dropping Sequences

The Oracle database provides a database object known as a Sequence. Sequences are used to automatically generate a series of unique numbers such as those used for Employee Id or Part Number columns. Sequences are not part of the ANSI SQL-92 standard. In the following example, an Oracle Sequence for Employee Id is created. The numbers to be generated will be between 1001 and 9999. As a rule of thumb, sequences can be named with the suffix `seq` to differentiate them from other database objects.

```
      CREATE SEQUENCE department_number_seq
      START WITH   1
      MAXVALUE     9999
      NOCYCLE ;

Sequence created.
```

In this example, the sequence will begin its numbering at 1 and count up (in increments of 1 which is the default) until it reaches 9999. Once the MAXVALUE is reached, accessing the sequence will return an error.

Sequences are accessed using a SELECT statement with a special table called `DUAL`. The DUAL table is a placeholder that exists in all schemas by default. In the following example, the next value in the `employee_id_seq` sequence is retrieved:

```
SELECT department_number_seq.nextval FROM dual ;

  NEXTVAL
---------
     6
```

Sequences can also be used in INSERT statements to automatically provide the next value for a key. For example, to insert a new employee row with the next employee id in the sequence, the following statement would be issued:

```
     INSERT INTO department VALUES
       (department_number_seq.nextval, 'Finance',
        '123456789', '01'-JAN-90');

    1 Row Created.
```

As with most database objects, Oracle Sequences can be dropped using a DROP SEQUENCE command. Dropping a sequence and then re-creating it has the effect of resetting the sequence to its START WITH number. In the following example, the Employee Id sequence created previously is dropped.

```
DROP SEQUENCE department_number_seq ;

Sequence dropped.
```

Sequences can also be altered to change the INCREMENT BY, MAXVALUE or START WITH values. The ALTER SEQUENCE statement achieves these changes.

Exercise 4: Working with Sequences

For this exercise, start by creating an Oracle Sequence called `student_id_seq`. Have the sequence start numbering at 120 and go up to 999.

Then, write an SQL INSERT statement to insert a new record for the following person:

```
Name:     Joe
Major:    CIS
GPA:      3.85
TutorID:  103
```

Use the `student_id_seq.nextval` as the StudentID. Finally, use a SELECT statement to query the V_CIS_MAJOR and see if the record was inserted properly.

4.2.2 Grant and Revoke Statements

The GRANT and REVOKE statements allow a user to control access to objects (Tables, Views, Sequences, Procedures, etc.) in their schema. The Grant command grants authorization for a *subject* (another user or group) to perform some *action* (SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX) on an *object* (Table, View, stored procedure, sequence or synonym).

The actions are defined as follows:

- **SELECT** - allows a subject to select rows from the object.
- **INSERT** - allows a subject to insert rows into the object.
- **UPDATE** - allows a subject to update rows in the object.
- **DELETE** - allows a subject to delete rows from the object.
- **ALTER** - allows a subject to alter the object. For example, add a column or change a constraint.
- **INDEX** - allows a subject to create an index on the object.
- **EXECUTE** - allows a subject to execute a stored procedure or trigger.

In addition to objects such as tables, the SELECT and UPDATE actions may also be granted on individual columns in a table or view.

The general syntax for the GRANT statement is:

```
GRANT , , ...
ON    tablename
TO    subject;
```

For example, assume user ALICE wishes to allow another user BOB to view the rows in the employee table. ALICE would execute the following GRANT statement:

```
GRANT SELECT
ON    employee
TO    BOB ;
```

At this point, user BOB may now issue SQL SELECT statements on the table `ALICE.employee`. For example, user BOB may execute:

```
SELECT * FROM ALICE.employee ;

FNAME    MI LNAME          SSN
-------- -- -------- ---------
JOHN     B  SMITH    123456789
FRANKLIN T  WONG     333445555
ALICIA   J  ZELAYA   999887777
JENNIFER S  WALLACE  987654321
RAMESH   K  NARAYAN  666884444
JOYCE    A  ENGLISH  453453453
AHMAD    V  JABBAR   987987987
JAMES    E  BORG     888665555
etc.
```

The REVOKE statement reverses the authorization by removing privileges from a subject (user). The syntax for REVOKE is:

```
REVOKE
ON
```