**XJTLU Entrepreneur College (Taicang)**

***School of* *AI and Advanced Computing***

**Lab Manual (Lab 4 and 5)**

***CPT103TC:***

***Introduction to Database***

Shaheen Khatoon

Second Part of Semester 1

**Academic Year 2021-2022**

**Preface**

This laboratory manual serves as supplemental material for the laboratory class. Majority of its contents were taken from the materials of Oracle’s DB Design and Programming with SQL training.

The table below shows the delivery plan with reference material.

|  |  |  |
| --- | --- | --- |
| **Labs** | **Topics** | **Reference in DB Programming with SQL** |
| **1** | Introduction to Oracle Application Express  SQL Workshop for uploading and running scripts | Lab set-up  Oracle Application Development Foundation (Self-study)  SQL Scripts |
| **2** | Demonstrate and end to end application building process | Oracle Application Development Foundation (Self-study)  Project OracleFlix-demo |
| **3** | Data modeling using SQL Developer and ER Assistant | SQL Developer and ER Assistant Tutorial |
| **4** | SQL DDL command to create database objects and constraints | Demo CompanyScript.SQl; |
| **5** | Managing constraints and SQL DML | Lab handouts |
| **6** | Basic SQL | Lab handouts |
| **7** | SQL Group function and subqueries | Lab handouts |
| **8** | SQL Joins | Lab handouts |
| **9** | SQL Single Row functions | Lab handouts |
| **10** | Application development | Lab handouts |

Table of Contents

[**Lab 4** 1](#_Toc95515467)

[**An Introduction to SQL** 1](#_Toc95515468)

[**Data Definition Language (DDL) commands in RDBMS** 1](#_Toc95515469)

[Oracle Datatypes 3](#_Toc95515470)

[**Lab Tasks** 4](#_Toc95515471)

[Create table Syntaxes and Examples 5](#_Toc95515472)

[Create tables for Company Database 5](#_Toc95515473)

[Try It / Solve It 1. 8](#_Toc95515474)

[ALTER TABLE 8](#_Toc95515475)

[DROP TABLE 10](#_Toc95515476)

[TRUNCATE 11](#_Toc95515477)

[Lab Exercise: 11](#_Toc95515478)

[**Intro to Constraints; NOT NULL and UNIQUE Constraints** 12](#_Toc95515479)

[**Creating Constraints** 12](#_Toc95515480)

[**Lab Exercise:** 13](#_Toc95515481)

[**PRIMARY KEY, FOREIGN KEY, and CHECK Constraints** 14](#_Toc95515482)

[PRIMARY KEY Constraints 14](#_Toc95515483)

[FOREIGN KEY (REFERENTIAL INTEGRITY) Constraints 14](#_Toc95515484)

[CHECK Constraints 17](#_Toc95515485)

[**Lab 5** 18](#_Toc95515486)

[**Managing Constraints** 18](#_Toc95515487)

[Adding Constraints 18](#_Toc95515488)

[Dropping Constraints 18](#_Toc95515489)

[Disabling Constraints 19](#_Toc95515490)

[Enabling Constraints 19](#_Toc95515491)

[Viewing Constraints 20](#_Toc95515492)

[**Data Manipulation Language (DML)** 20](#_Toc95515493)

[INSERT Statements Syntax and Examples 21](#_Toc95515494)

[Updating Column Values and Deleting Rows 22](#_Toc95515495)

[DELETE Syntax and Examples 23](#_Toc95515496)

[**Integrity Constraint Errors** 24](#_Toc95515497)

[**Lab Exercise:** 25](#_Toc95515498)

[Insert Operation 26](#_Toc95515499)

[Delete Operation 27](#_Toc95515500)

[Update Operation 27](#_Toc95515501)

# **Lab 4**

# **An Introduction to SQL**

SQL or Structured Query Language is a programming language designed to manage or communicate with a database.

Types of SQL include:

* Data Definition Language (DDL)
* Data Manipulation Language (DML)
* Transaction Control Statements
* Session Control Statements
* System Control Statements
* Embedded SQL Statement

In this module we will only focus on DDL and DML

Other SQL types will be covered in the next module (Database Design and Development)

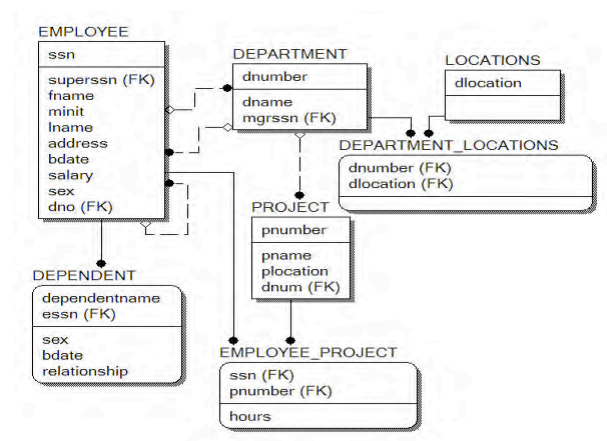
# **Data Definition Language (DDL) commands in RDBMS**

Some DDL statements include:

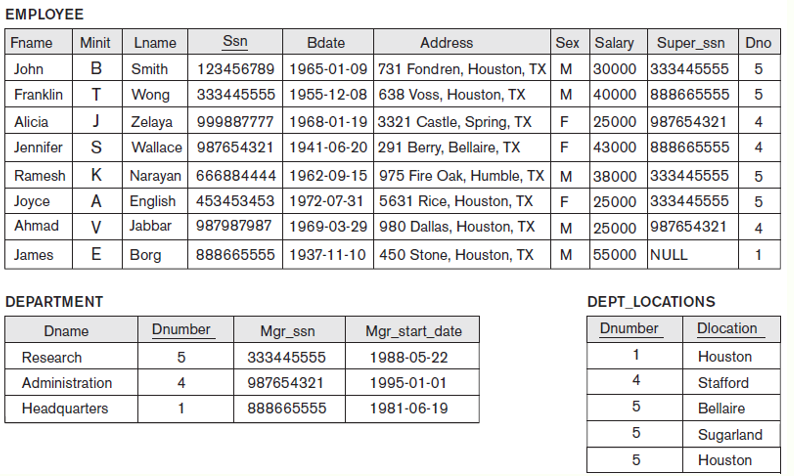
|  |  |  |
| --- | --- | --- |
| ALTER SEQUENCE | CREATE PROCEDURE | DROP PROCEDURE |
| ALTER SESSION | CREATE SCHEMA | DROP SCHEMA |
| ALTER TABLE | CREATE SEQUENCE | DROP SEQUENCE |
| ALTER TRIGGER | CREATE SYNONYM | DROP SYNONYM |
| ALTER USER | CREATE TABLE | DROP TABLE |
| ALTER VIEW | CREATE TRIGGER | DROP TRIGGER |
| CREATE DATABASE | CREATE USER | DROP USER |
| CREATE FUNCTION | CREATE VIEW | DROP VIEW |
| CREATE GLOBAL TEMPORARY TABLE | DROP FUNCTION | GRANT |
| CREATE INDEX | DROP INDEX | REVOKE |
| CREATE JAVA | DROP JAVA | TRUNCATE TABLE |

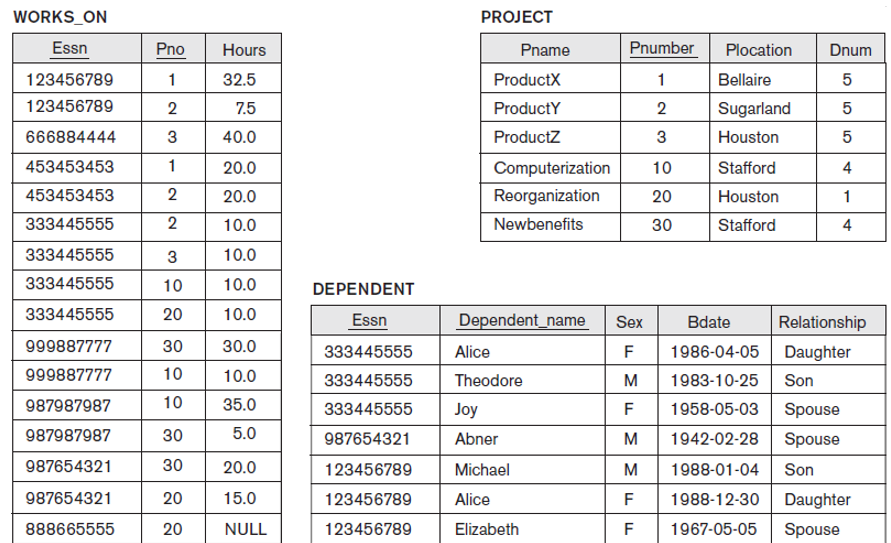
We will use Company case study to demonstrate main concept during the lab. Students need to practice each lab using Company databse project exercises provided along this manual.

An higher level design of company database is shown in the below figure.



The company database state is shown in the following figure:





## Oracle Datatypes

Before creating database object, take few minutes to look at Oracle datatypes

Built-in datatypes in Oracle can be divided into six categories: character, numeric, DATE, LOB, RAW and LONG RAW, and ROWID and UROWID. We will concentrate only on the first three. Please refer to the tables below.

**Character Datatypes**

|  |  |
| --- | --- |
| **Datatype** | **Explanation** |
| CHAR | It stores a fixed length of characters between 1 and 2000 bytes. For example if your column Name was defined as CHAR(20) a value “Ali” will still be of length 20 with 17 padded spaces. |
| VARCHAR2, VARCHAR | They store a variable length of characters between 1 and 4000 bytes. Using the same example above but with Name column defined as VARCHAR2(20), “Ali” will be only of length 3.  The only difference between VARCHAR and VARCHAR2 is how they see NULL and empty string. The first one treats them differently while the latter as the same.  It is suggested that we use VARCHAR2. |
| NCHAR, NVARCHAR2 | They store Unicode characters. NCHAR can hold 2000 bytes while NVARCHAR2 can extend up to 4000 bytes. |

**Numeric Datatypes**

|  |  |
| --- | --- |
| **Datatype** | **Explanation** |
| NUMBER | It stores fixed and floating-point numbers up to 38 digits of precision. You may optionally define its precision (total no. of digits) and scale (no. of digits on the right). For example Grade column can be defined as simply NUMBER or NUMBER(4,2). |
| BINARY\_FLOAT | It stores a 32-bit, single-precision floating-point number datatype. |
| BINARY\_DOUBLE | It stores a 64-bit, double-precision floating-point number datatype. |

**DATE** datatype stores both date and time values. It specifically stores the year, month, day, hours, minutes, and seconds. Most of the time, we use the TO\_DATE function to store a date value. Look at the examples below.

|  |  |
| --- | --- |
| **Example** | **Explanation** |
| TO\_DATE(‘March 18, 1990’, ‘MONTH DD, YYYY’) | Stores the date with all zeroes for hours, minutes and seconds |
| TO\_DATE(’18-MAR-90 12:45 A.M.’,  ‘DD-MON-YY HH:MI A.M.’) | Stores a complete set of date and time. |

## **Lab Tasks**

In this lab you will create tables and insert data into tables. Use Application express to practice each example shown in the query code and observe results.

Table Creation Rules

* All data in a relational database is stored in tables
* When creating a new table, use the following rules for table names and column names:
* Must begin with a letter −Must be 1 to 30 characters long
* Must contain only A - Z, a - z, 0 - 9, \_ (underscore), $, and # −Must not duplicate the name of another object owned by the same user
* Must not be an Oracle Server reserved word

### Create table Syntaxes and Examples

Create table command defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

CREATE TABLE table

<col1> <data type> (<size>)>,…<coln> <data type> (<size>)>;

CREATE TABLE my\_cd\_collection

(cd\_number NUMBER(3),

title VARCHAR2(20),

artist VARCHAR2(20),

purchase\_date DATE DEFAULT SYSDATE);

CREATE TABLE my\_friends

(first\_name VARCHAR2(20),

last\_name VARCHAR2(30),

email VARCHAR2(30),

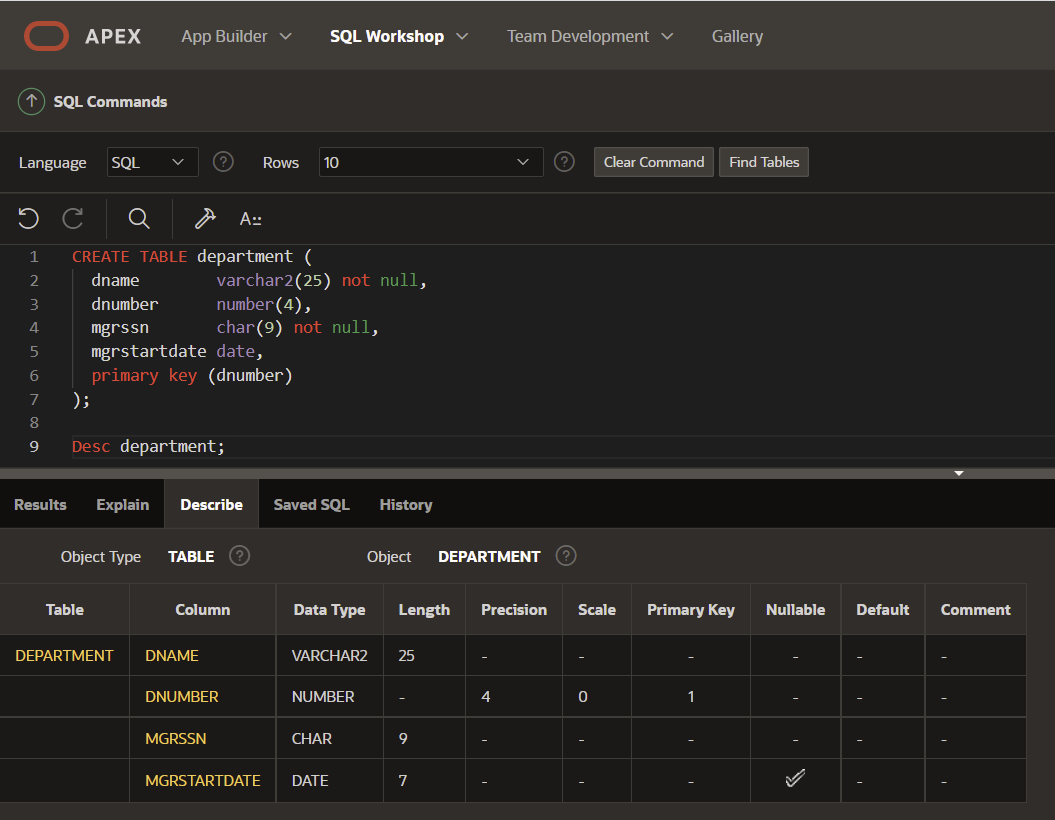
phone\_num VARCHAR2(12),

birth\_date DATE);

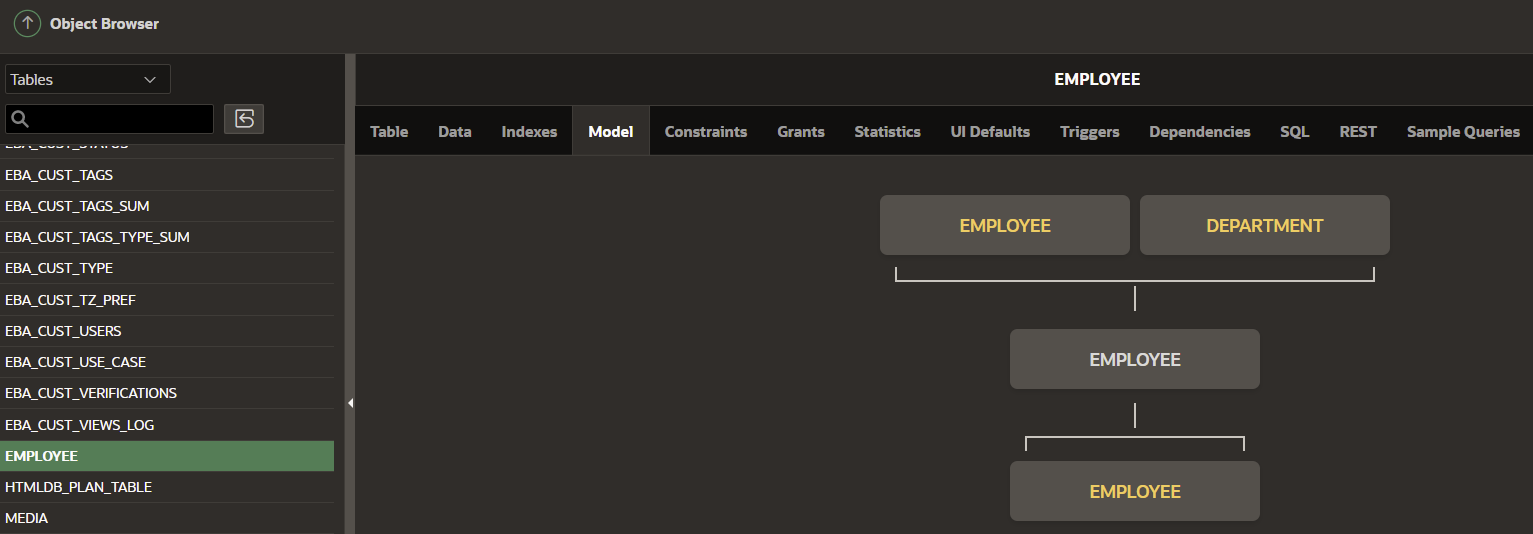
### Create tables for Company Database

|  |  |
| --- | --- |
| Example (company database) | Explanation |
| CREATE TABLE department (  dname varchar2(25) not null,  dnumber number(4),  mgrssn char(9) not null,  mgrstartdate date,  primary key (dnumber)  ); | A table named **department** was created with 4 columns.  Note: You can use DESCRIBE or its abbreviated form DESC to view table descriptions.  Syntax: DESCRIBE<*tablename*>;  DESCRIBE department; |
| CREATE TABLE employee (  fname varchar2(15) not null,  minit varchar2(1),  lname varchar2(15) not null,  ssn char(9),  bdate date,  address varchar2(50),  sex char,  salary number(10,2),  superssn char(9),  dno number(4),  primary key (ssn),  foreign key (superssn) references employee(ssn),  foreign key (dno) references department(dnumber)  ); | Note: you must create department table first as it is used in the employee table as foreign key. However, you can create employee table first by skipping the department foreign key. Once department table is created you can update employee table by using AlTER command statements describe below.  Number(10, 2) 10表示总长度 2 表示小数点位数 |
| CREATE TABLE dept\_locations (  dnumber number(4),  dlocation varchar2(15),  primary key (dnumber,dlocation),  foreign key (dnumber) references department(dnumber)  ); |  |
| CREATE TABLE project (  pname varchar2(25) not null,  pnumber number(4),  plocation varchar2(15),  dnum number(4) not null,  primary key (pnumber),  unique (pname),  foreign key (dnum) references department(dnumber)  ); |  |
| CREATE TABLE works\_on (  essn char(9),  pno number(4),  hours number(4,1),  primary key (essn,pno),  foreign key (essn) references employee(ssn),  foreign key (pno) references project(pnumber)  ); |  |

A screenshot of creating department table is shown in the below figure:

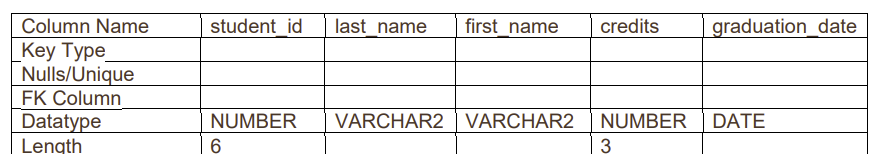


After creating employee table go to object browser and inspect employee table to see how foreign key allow connecting employee and department tables. Also inspect other properties such as constraints etc.



### Try It / Solve It 1.

Complete the GRADUATE CANDIDATE table instance chart. Credits is a foreign-key column referencing the requirements table.



Write the syntax to create the grad\_candidates table.

Confirm creation of the table using DESCRIBE.

## ALTER TABLE

* ALTER TABLE statements are used to:
  + Add a new column
  + Modify an existing column
  + Define a DEFAULT value for a column
  + Drop a column
* DROP TABLE
* RENAME
* TRUNCAT

• You can add or modify a column in a table, but you cannot specify where the column appears

**ALTER TABLE: Adding a Column**

• To add a new column, use the SQL syntax :

*ALTER TABLE tablename*

*ADD (column name data type [DEFAULT expression],*

*column name data type [DEFAULT expression], ...*

Examples:

ALTER TABLE my\_cd\_collection

ADD (release\_date DATE DEFAULT SYSDATE);

ALTER TABLE my\_friends

ADD (favorite\_game VARCHAR2(30));

**ALTER TABLE: Modifying a Column**

Modifying a column can include changes to a column's data type, size, and DEFAULT value

Rules and restrictions when modifying a column are:

* You can increase the width or precision of a numeric column
* You can increase the width of a character column
* You can decrease the width of a NUMBER column if the column contains only null values, or if the table has no rows
* For VARCHAR types, you can decrease the width down to the largest value contained in the column
* You can change the data type only if the column contains null values
* You can convert a CHAR column to VARCHAR2 or convert a VARCHAR2 COLUMN to CHAR only if the
* column contains null values, or if you do not change the size to something smaller than any value in the
* column
* A change to the DEFAULT value of a column affects only later insertions to the table

**ALTER TABLE: Modifying a Column Example**

Example: a table has been created with two columns:

CREATE TABLE mod\_emp

(last\_name VARCHAR2(20),

salary NUMBER(8,2));

ALTER TABLE mod\_emp

MODIFY (last\_name VARCHAR2(10)); Would be permitted only if columns were empty, or the largest name was 10 or less characters

ALTER TABLE mod\_emp

MODIFY (last\_name VARCHAR2(30)); Modification would be permitted with or without data as column width increased

ALTER TABLE mod\_emp

MODIFY (salary NUMBER(8,2) DEFAULT 50); Would be permitted with or without data as only a DEFAULT value added

**ALTER TABLE: Dropping a Column**

When dropping a column the following rules apply:

−A column containing data may be dropped

−Only one column can be dropped at a time

−You can't drop all of the columns in a table; at least one column must remain

−Once a column is dropped, the data values in it cannot be recovered

SQL Syntax: ALTER TABLE tablename

DROP COLUMN column name;

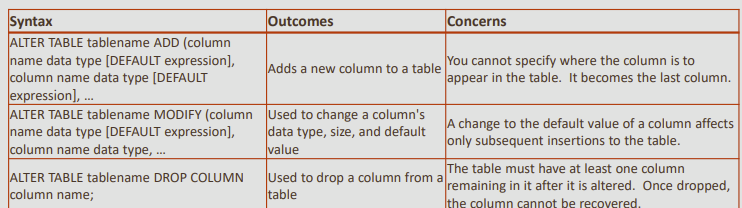
ALTER TABLE my\_cd\_collection

DROP COLUMN release\_date;

ALTER TABLE my\_friends

DROP COLUMN favorite\_game;

This chart summarizes the uses of the ALTER TABLE command:



## DROP TABLE

The DROP TABLE statement removes the definition of an Oracle table

The database loses all the data in the table and all the indexes associated with it

When a DROP TABLE statement is issued:

* All data is deleted from the table
* The table's description is removed from the Data Dictionary

The Oracle Server does not question your decision and it drops the table immediately

**Syntax**:

ALTER SESSION SET RECYCLEBIN=ON;

DROP TABLE tablename;

Example:

ALTER SESSION SET RECYCLEBIN=ON;

DROP TABLE copy\_employees;

**RENAME TABLE**

To change the name of a table, use the RENAME statement

This can be done only by the owner of the object or by the DBA

• Syntax: **RENAME old\_name to new\_name;**

• Example: **RENAME my\_cd\_collection TO my\_music;**

•We will see later that we can rename other types of

## TRUNCATE

• Truncating a table removes all rows from a table and releases the storage space used by that table

•When using the TRUNCATE TABLE statement:

−You cannot roll back row removal

−You must be the owner of the table or have been given DROP ANY TABLE system privileges

**Syntax**: **TRUNCATE TABLE tablename;**

## Lab Exercise:

Before beginning the practice exercises, execute a DESCRIBE for each of the following tables:

o\_employees, o\_departments and o\_jobs. These tables will be used in the exercises. If they do not

exist in your account, create them as follows:

1. Create the three o\_tables – jobs, employees, and departments – using the syntax:

CREATE TABLE o\_jobs AS (SELECT \* FROM jobs);

CREATE TABLE o\_employees AS (SELECT \* FROM employees);

CREATE TABLE o\_departments AS (SELECT \* FROM departments);

2. Add the Human Resources job to the jobs table:

INSERT INTO o\_jobs (job\_id, job\_title, min\_salary, max\_salary)

VALUES('HR\_MAN', 'Human Resources Manager', 4500, 5500);

3. Add the three new employees to the employees table:

INSERT INTO o\_employees (employee\_id, first\_name, last\_name, email, hire\_date,

job\_id)

VALUES(210, 'Ramon', 'Sanchez', 'RSANCHEZ', SYSDATE, 'HR\_MAN');

4. Add Human Resources to the departments table:

INSERT INTO o\_departments(department\_id, department\_name)

VALUES (210,'Human Resources');

# **Intro to Constraints; NOT NULL and UNIQUE Constraints**

The five types of CONSTRAINT are:

1. NOT NULL constraints
2. UNIQUE constraints
3. PRIMARY KEY constraints
4. FOREIGN KEY constraints
5. CHECK constraints

## **Creating Constraints**

* Recall the SQL syntax for creating a table
* In the CREATE TABLE statement session, each column and its data type is defined
* You use the CREATE TABLE statement to establish constraints for each column in the table
* There are two different places in the CREATE TABLE statement that you can specify the constraint details:
* At the column level next to the name and data type
* At the table level after all the column names are listed

**Column level Constraints**

CREATE TABLE clients

(client\_number NUMBER(4) CONSTRAINT clients\_cient\_num\_pk PRIMARY KEY,

last\_name VARCHAR2(13) CONSTRAINT clients\_last\_name\_nn NOT NULL,

email VARCHAR2(80) CONSTRAINT clients\_emil\_uk UNIQUE);

According to naming convention:

−A primary key constraint on client\_number would be named clients\_client\_number\_pk

−A not null constraint on last\_name would be named clients\_last\_name\_nn

−A unique constraint on email address would be named clients\_email\_uk

**Table level Constraints**

CREATE TABLE clients(

client\_number NUMBER(6),

first\_name VARCHAR2(20),

last\_name VARCHAR2(20),

phone VARCHAR2(20) CONSTRAINT NOT NULL,

email VARCHAR2(10) CONSTRAINT NOT NULL,

CONSTRAINT phone\_email\_uk UNIQUE(email,phone),

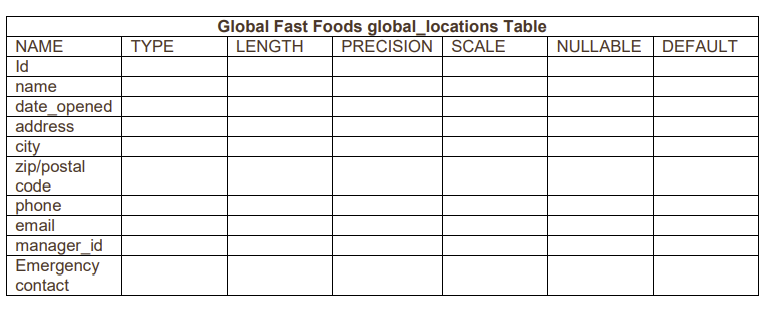
CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number));

**Basic Rules for Constraints**

* Constraints that refer to more than one column (a composite key) must be defined at the table level
* The NOT NULL constraint can be specified only at the column level, not the table level
* UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK constraints can be defined at either the column or table level
* If the word CONSTRAINT is used in a CREATE TABLE statement, you must give the constraint a name

## **Lab Exercise:**

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store’s locations. The owners want to make sure that all entries have an identification number, date opened, address, and city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global\_locations table. Use the table for your answers.



Based on the information provided by the owners,

1. choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.
2. Use “nullable” to indicate those columns that can have null values.
3. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.
4. Execute the CREATE TABLE statement in Oracle Application Express.
5. Execute a DESCRIBE command to view the Table Summary information.
6. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement

# **PRIMARY KEY, FOREIGN KEY, and CHECK Constraints**

## PRIMARY KEY Constraints

* A PRIMARY KEY constraint is a rule that the values in one column or a combination of columns must uniquely identify each row in a table
* No primary-key value can appear in more than one row in the table
* To satisfy a PRIMARY KEY constraint, both of the following conditions must be true:
  + No column that is part of the primary key can contain a null
  + A table can have only one primary key

In a CREATE TABLE statement, the column-level PRIMARY KEY constraint syntax is stated:

CREATE TABLE clients

(client\_number NUMBER(4) CONSTRAINT clients\_client\_num\_pk PRIMARY KEY,

first\_name VARCHAR2(14),

last\_name VARCHAR2(13));

To create the PRIMARY KEY constraint at table-level the syntax is:

CREATE TABLE clients

(client\_number NUMBER(4),

first\_name VARCHAR2(14),

last\_name VARCHAR2(13),

CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number));

Note that the PRIMARY KEY column name follows the constraint type, and is enclosed in parenthesis

## FOREIGN KEY (REFERENTIAL INTEGRITY) Constraints

* FOREIGN KEY constraints are also called "referential integrity" constraints
* Foreign Key constraints designate a column or combination of columns as a foreign key
* A foreign keys links back to the primary key (or a unique key) in another table, and this link is the basis of the relationship between tables
* The table containing the foreign key is called the "child" table and the table containing the referenced key is called the "parent" table



* In the tables shown, the primary-key of the DEPARTMENTS table, department\_id, also appears in the EMPLOYEES table as a foreign-key column
* satisfy a referential-integrity constraint, a foreign key value must match an existing value in the parent table or be NULL
* A primary-key value can exist without a corresponding foreign-key value; however, a foreign-key must have a corresponding primary key
* The rule is: before you define a referential-integrity constraint in the child table, the referenced UNIQUE or PRIMARY KEY constraint on the parent table must already be defined
* In other words, you must first have a parent primary key defined before you can create a foreign key in a child table

The syntax for defining a FOREIGN KEY constraint requires a reference to the table and column in the parent table

* A FOREIGN KEY constraint in a CREATE TABLE statement can be defined as follow

CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

first\_name VARCHAR2(20),

last\_name VARCHAR2(25),

department\_id NUMBER(4,0) CONSTRAINT c\_emps\_dept\_id\_fk REFERENCES departments(department\_id),

email VARCHAR2(25));

NOTE: when defining a foreign key at column level the words "foreign key" are NOT included.

* Table-level syntax example:

CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

first\_name VARCHAR2(20),

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

email VARCHAR2(25),

CONSTRAINT c\_emps\_dept\_id\_fk FOREIGN KEY (department\_id)

REFERENCES departments(department\_id));

NOTE: when defining a foreign key at table level the words "foreign key" ARE included before the name of the table and column that has the constraint.

**ON DELETE CASCADE** - Maintaining Referential Integrity

Using the ON DELETE CASCADE option when defining a foreign key enables the dependent rows in the child table to be deleted when a row in the parent table is deleted

If the foreign key does not have an ON DELETE CASCADE option, referenced rows in the parent table cannot be deleted

In other words, the child table FOREIGN KEY constraint includes the ON DELETE CASCADE permission allowing its parent to delete the rows that it refers to

In the above example attempt to delete a department e.g. department\_id 110 from the departments table fails as there are dependent rows in the employee table

Table created with ON DELETE CASCADE:

CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

first\_name VARCHAR2(20),

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

email VARCHAR2(25),

CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id)

REFERENCES copy\_departments(department\_id) ON DELETE CASCADE);

cascade 还是null 加到后面就可以了

An attempt to delete department\_id 110 from the departments table succeeds, and the dependent rows in the employee table are also deleted

ON DELETE SET NULL

Rather than having the rows in the child table deleted when using an ON DELETE CASCADE option, the child rows can be filled with null values using the ON DELETE SET NULL option

CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

first\_name VARCHAR2(20),

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

email VARCHAR2(25),

CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id)

REFERENCES copy\_departments(department\_id) ON DELETE SET NULL);

## CHECK Constraints

The CHECK constraint explicitly defines a condition that must be met

To satisfy the constraint, each row in the table must make the condition either True or unknown (due to a null)

The condition of a CHECK constraint can refer to any column in the specified table, but not to columns of other tables

**CHECK Constraint Example:**

This CHECK constraint ensures that a value entered for end\_date is later than start\_date

CREATE TABLE copy\_job\_history

(employee\_id NUMBER(6,0),

start\_date DATE,

end\_date DATE,

job\_id VARCHAR2(10),

department\_id NUMBER(4,0),

CONSTRAINT cjhist\_emp\_id\_st\_date\_pk

PRIMARY KEY(employee\_id, start\_date),

CONSTRAINT cjhist\_end\_ck CHECK (end\_date > start\_date));

As this CHECK CONSTRAINT is referencing two columns in the table, it MUST be defined at table level

CHECK constraints can be defined at the column level or the table level

Column-level syntax:

salary NUMBER(8,2) CONSTRAINT employees\_min\_sal\_ck CHECK (salary > 0)

Table-level syntax:

CONSTRAINT employees\_min\_sal\_ck CHECK (salary > 0)

# **Lab 5**

# **Managing Constraints**

The ALTER TABLE statement is used to make changes to constraints in existing tables

These changes can include adding or dropping constraints, enabling or disabling constraints, and adding a NOT NULL constraint to a column

The guidelines for making changes to constraints are:

* You can add, drop, enable, or disable a constraint, but you cannot modify its structure
* You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE statement
* MODIFY is used because NOT NULL is a column-level change.
* You can define a NOT NULL constraint only if the table is empty or if the column contains a value for every row

## Adding Constraints

To add a constraint to an existing table, use the following SQL syntax:

ALTER TABLE table\_name

ADD [CONSTRAINT constraint\_name] type of constraint(column\_name);

If the constraint is a FOREIGN KEY constraint, the REFERENCES keyword must be included in the

statement

ALTER TABLE tablename

ADD CONSTRAINT constraint\_name FOREIGN KEY(column\_name) REFERENCES tablename(column\_name);

The following example demonstrates the syntax to add this foreign key to the EMPLOYEES table:

ALTER TABLE employees

ADD CONSTRAINT emp\_dept\_fk FOREIGN KEY (department\_id)

REFERENCES departments (department\_id) ON DELETE CASCADE;

If the constraint is a NOT NULL constraint, the ALTER TABLE statement uses MODIFY in place of ADD

NOT NULL constraints can be added only if the table is empty or if the column contains a value for every row:

ALTER TABLE table\_name

MODIFY (column\_name CONSTRAINT constraint\_name NOT NULL);

ALTER TABLE employees

MODIFY (email CONSTRAINT emp\_email\_nn NOT NULL);

## Dropping Constraints

* To drop a constraint, you need to know the name of the constraint
* If you do not know it, you can find the constraint name from the USER\_CONSTRAINTS and USER\_CONS\_COLUMNS in the data dictionary
* The CASCADE option of the DROP clause causes any dependent constraints also to be dropped. To drop the primary-key constraint on the DEPARTMENTS table, the word CASCADE is used to drop the foreign-key constraint in the child table. So if the PRIMARY KEY constraint in the departments table is dropped with the CASCADE option, the FOREIGN KEY constraint in the employees table would also be dropped. 也就是说只要PK被dropped了 那么与之又联系的FK也会断掉
* Note that when you drop an integrity constraint, that constraint is no longer enforced by the Oracle Server and is no longer available in the data dictionary

ALTER TABLE table\_name

DROP CONSTRAINT name [CASCADE]

ALTER TABLE copy\_departments

DROP CONSTRAINT c\_dept\_dept\_id\_pk CASCADE;

## Disabling Constraints

You can disable a constraint without dropping it or recreating it by using the ALTER TABLE option DISABLE

DISABLE allows incoming data, whether or not it conforms to the constraint

This function allows data to be added to a child table without having corresponding values in the parent

table

DISABLE simply switches off the constraint

You can use the DISABLE clause in both the ALTER TABLE statement and the CREATE TABLE statemen

Disabling a unique or primary-key constraint removes the unique index

CREATE TABLE copy\_employees

( employee\_id NUMBER(6,0) PRIMARY KEY DISABLE,

...

...);

ALTER TABLE copy\_employees

DISABLE CONSTRAINT c\_emp\_dept\_id\_fk

The CASCADE clause disables dependent integrity constraints. If the constraint is later enabled, the

dependent constraints are not automatically enabled

Syntax and example:

ALTER TABLE table\_name

DISABLE CONSTRAINT constraint\_name [CASCADE];

ALTER TABLE copy\_departments

DISABLE CONSTRAINT c\_dept\_dept\_id\_pk CASCADE;

## Enabling Constraints

To activate an integrity constraint currently disabled, use the ENABLE clause in the ALTER TABLE statement

ENABLE ensures that all incoming data conforms to the constraint

Syntax and example

ALTER TABLE table\_name

ENABLE CONSTRAINT constraint\_name;

ALTER TABLE copy\_departments

ENABLE CONSTRAINT c\_dept\_dept\_id\_pk;

You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement

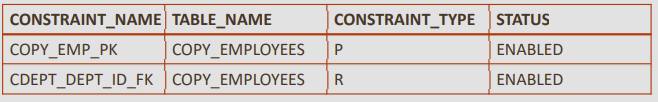
## Viewing Constraints

To view all constraints on your table, query the USER\_CONSTRAINTS table

SELECT constraint\_name, table\_name, constraint\_type, status

FROM USER\_CONSTRAINTS

WHERE table\_name ='COPY\_EMPLOYEES'; 必须要大写



• The constraint types listed in the Data Dictionary are:

P – PRIMARY KEY;

R – REFERENCES (foreign key);

C – CHECK constraint (including NOT NULL);

U – UNIQUE

# **Data Manipulation Language (DML)**

CRUD (Create Read Update Delete) operations are 4 basic actions that can be applied to a database object. In this section, we will create a row (using INSERT), read a row (using SELECT), update a row (using UPDATE), and delete a row (using DELETE).

**Copy Tables Before Inserting**

To keep your schema tables in their original state, you will make a copy of each table before completing the practice activities in this and later lessons

In each practice activity, you will use the copy of the table that you create, not the original

If you inadvertently alter a table copy, you can use the original table to restore the copy

You should name each copied table: copy\_tablename

The table copies will not inherit the associated primary-to-foreign-key integrity rules (relationship constraints) of the original tables

The column data types, however, are inherited in the copied tables

CREATE TABLE *copy\_tablename* AS (SELECT \* FROM *tablename*);

|  |  |
| --- | --- |
| Example | Explanation |
| CREATE TABLE copy\_employees AS (SELECT \* FROM employees);  CREATE TABLE copy\_departments AS (SELECT \* FROM departments); | Following two tables were created:  copy\_employees  copy\_departments  With all the attributes from the original employees and departments tables |
| DESCRIBE copy\_employees;  SELECT \* FROM copy\_employees;  DESCRIBE copy\_departments;  SELECT \* FROM copy\_departments; | To verify and view the copy of the table, use the DESCRIBE and SELECT statements: |

## INSERT Statements Syntax and Examples

The INSERT statement is used to add a new row to a table.

The statement requires three values:

1. the name of the table
2. the names of the columns in the table to populate
3. corresponding values for each column

•How can we INSERT the data below to create a new department in the copy\_departments table?

|  |  |
| --- | --- |
| Example | Explanation |
| INSERT INTO copy\_departments (department\_id, department\_name, manager\_id, location\_id)  VALUES (200,'Human Resources', 205, 1500); | This statement explicitly **lists each column** as it appears in the table  The values for each column are listed in the same order  −Note that number values are not enclosed in single quotation marks |
| INSERT INTO copy\_departments  VALUES (210,'Estate Management', 102, 1700); | Another way to insert values in a table is to implicitly add them by **omitting the column** names  **One precaution:** the values for each column must match exactly the default order in which they appear in the table (as shown in a DESCRIBE statement), and a value must be provided for each column  For clarity, however, it is best to use the column names in an INSERT clause |
| INSERT INTO sales\_reps(id, name, salary, commission\_pct)这些column都是要在table中创建好的  SELECT employee\_id, last\_name, salary, commission\_pct  FROM employees  WHERE job\_id LIKE '%REP%'; | **Using A Subquery To Copy Rows**  All the results from the subquery are inserted into the table  So we can copy 100 rows – or 1000 rows – with one multiple-row subquery within the INSERT  As you would expect, you don't need a VALUES clause when using a subquery to copy rows because the inserted values will be exactly the values returned by the subquery  In the example a new table called SALES\_REPS is being populated with copies of some of the rows and columns from the EMPLOYEES table  The WHERE clause is selecting those employees that have job IDs like '%REP%' |
| INSERT INTO sales\_reps SELECT \* FROM employees; | If we want to copy all the data – all rows and all columns – the syntax is even simpler To select all rows from the EMPLOYEES table and insert them into the SALES\_REPS table, the statement would be written as shown  Again, this will work only if both tables have the same number of columns with matching data types, and they are in the same order |

## Updating Column Values and Deleting Rows

UPDATE <*table\_name*>

SET <*column*> = <*value*>

WHERE <*condition*> ;

|  |  |
| --- | --- |
| Example | Explanation |
| UPDATE copy\_employees  SET phone\_number = '123456'  WHERE employee\_id = 303; | UPDATE statement to change the phone number of one employee in the employees table |
| UPDATE copy\_employees  SET phone\_number = '654321', last\_name = 'Jones'  WHERE employee\_id >= 303; |  |
| UPDATE copy\_employees  SET phone\_number = '654321', last\_name = 'Jones' | Take care when updating column values  If the WHERE clause is omitted, every row in the table will be updated  This may not be what was intended |
| UPDATE copy\_employees  SET salary = (SELECT salary  FROM copy\_employees  WHERE employee\_id = 100)  WHERE employee\_id = 101; | **Updating a Column with a value from a Subquery**  This example changes the salary of one employee (id = 101) to the same salary as another employee (id = 100)  As usual, the subquery executes first and retrieves the salary for employee id = 100 This salary value is then used to update the salary for employee id = 101 |
| UPDATE copy\_employees  SET salary = (SELECT salary  FROM copy\_employees  WHERE employee\_id = 205),  job\_id = (SELECT job\_id  FROM copy\_employees  WHERE employee\_id = 205)  WHERE employee\_id = 206; | **Updating Two Columns with Two Subquery Statements**  In the following example the UPDATE statement changes the salary and job id of one employee (id = 206) to the same values as another employee (id = 205) |

## DELETE Syntax and Examples

The DELETE statement is used to remove existing rows in a table

The statement requires two values:

1. the name of the table
2. the condition that identifies the rows to be deleted

DELETE FROM <*table\_name*>

WHERE <*condition*> ;

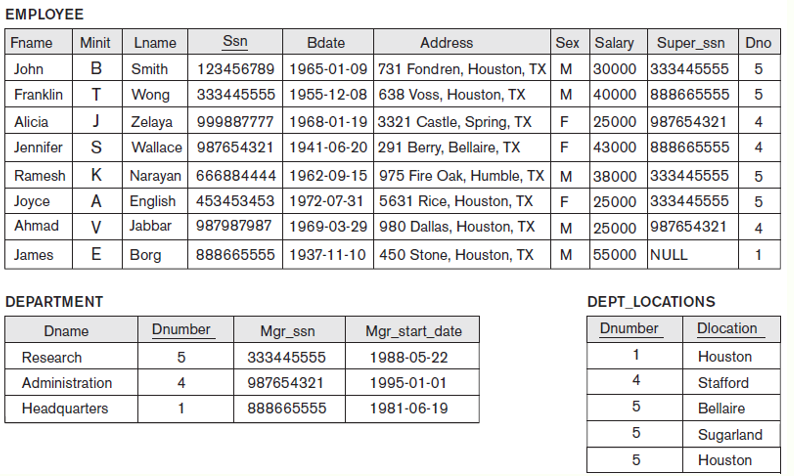
|  |  |
| --- | --- |
| Example | Explanation |
| DELETE from copy\_employees  WHERE employee\_id = 303; | The example shown uses the copy employee table to delete one row—the employee with ID number 303 |
| DELETE FROM copy\_employees; | All rows in the table are deleted if you omit the WHERE clause |
| DELETE FROM copy\_employees  WHERE department\_id =  (SELECT department\_id  FROM departments  WHERE department\_name = 'Shipping'); | **Subquery DELETE**  The example shown deletes rows from the employees table for all employees who work in the Shipping department |
| DELETE FROM copy\_employees e  WHERE e.manager\_id IN  (SELECT d.manager\_id  FROM employees d  HAVING count (d.department\_id) < 2  GROUP BY d.manager\_id); | deletes rows of all employees who work for a manager that manages less than 2 employees |
| UPDATE copy\_employees  SET last\_name = (SELECT last\_name  FROM copy\_employees  WHERE employee\_id = 999)  WHERE employee\_id = 101; | **Integrity Constraint Errors** Integrity constraints ensure that the data conforms to a needed set of rules  The constraints are automatically checked whenever a DML statement which could break the rules is executed  If any rule would be broken, the table is not updated and an error is returned  This example violates a NOT NULL constraint because last\_name has a not null constraint and id=999 does not exist, so the subquery returns a null result |
| 1. UPDATE employees SET department\_id = 15   WHERE employee\_id = 100;   1. DELETE FROM departments WHERE department\_id = 10; 2. UPDATE employees SET department\_id = 10 WHERE department\_id = 20; | When will primary key - foreign key constraints be checked?  The EMPLOYEES table has a foreign key constraint on department\_id which references the department\_id of the DEPARTMENTS table  This ensures that every employee belongs to a valid department  In the DEPARTMENTS table, department\_ids 10 and 20 exist but 15 does not  Query 1: Will return an error because department\_id 15 does not exist.  Query 2: Will return an error if there are employees with a department\_id of 10  Query 3: Will succeed assuming that department\_id 10 is in the departments table. |

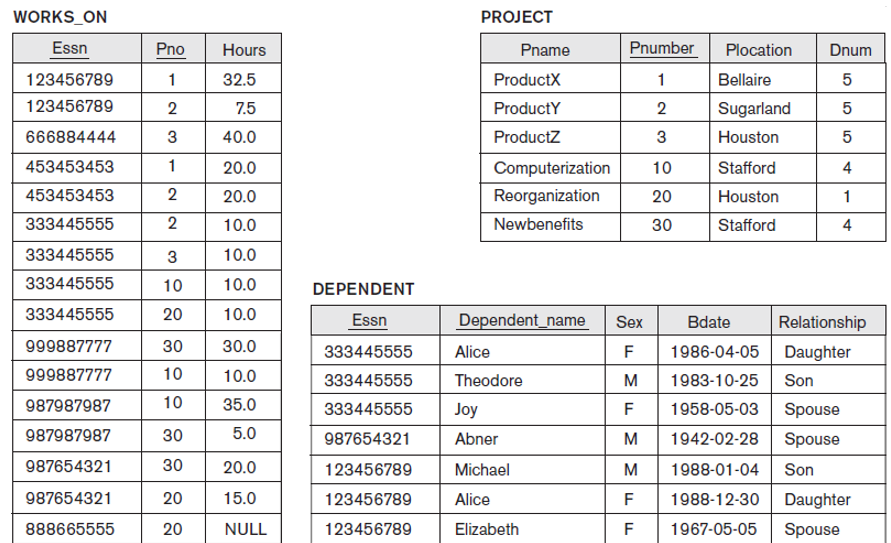
# **Lab Exercise:**

By now you have created company database tables. If not use create table statements from the previous section to create table.

Once table are created, populate them using the below database state to create complete database for company which will be used later for performing basic and advanced queries:

SQL script for the below database is also provided with lab.





Data load script is also available with this lab, you can use the script to create the company database:

Perform insert, update and delete operations on the above database state while watching for domain, integrity, key and referential integrity constraints. Make copy of each table if you like to keep the original data intact.

Here are the few examples, please formulate appropriate syntax to execute each operation in Application Express. Note your results might be different based on the database state.

## Insert Operation

**Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, NULL, ‘1960-04-05’, ‘6357 Windy Lane, Katy,**

**TX’, F, 28000, NULL, 4> into EMPLOYEE.**

**SQL Command:**

insert into employee values('Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4);

**Result:** This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it should be rejected.

For each of the below operations, write an appropriate query to verify your results either rejected due to violating any constraint or performed successfully.

**■ Operation:**

Insert <‘Alicia’, ‘J’, ‘Zelaya’, ‘999887777’, ‘1960-04-05’, ‘6357 Windy Lane, Katy, TX’, F, 28000, ‘987654321’, 4> into EMPLOYEE.

Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.

**■ Operation:**

Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, ‘677678989’, ‘1960-04-05’, ‘6357 Windswept, Katy, TX’, F, 28000, ‘987654321’, 7> into EMPLOYEE.

**Result:** This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with Dnumber = 7.

**■ Operation:**

Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, ‘677678989’, ‘1960-04-05’, ‘6357 Windy Lane,

Katy, TX’, F, 28000, NULL, 4> into EMPLOYEE.

**Result:** This insertion satisfies all constraints, so it is acceptable

## Delete Operation

The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted. Here are some examples.

■ Operation:

Delete the WORKS\_ON tuple with Essn = ‘999887777’ and Pno = 10.

**Result:** This deletion is acceptable and deletes exactly one tuple.

■ Operation:

Delete the EMPLOYEE tuple with Ssn = ‘999887777’.

**Result:** This deletion is not acceptable, because there are tuples in WORKS\_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

■ Operation:

Delete the EMPLOYEE tuple with Ssn = ‘333445555’.

**Result:** This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS\_ON, and DEPENDENT relations.

## Update Operation

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

**■ Operation:**

Update the salary of the EMPLOYEE tuple with Ssn = ‘999887777’ to 28000.

Result: Acceptable.

**■ Operation:**

Update the Dno of the EMPLOYEE tuple with Ssn = ‘999887777’ to 1.

Result: Acceptable.

**■ Operation:**

Update the Dno of the EMPLOYEE tuple with Ssn = ‘999887777’ to 7.

Result: Unacceptable, because it violates referential integrity.

**■ Operation:**

Update the Ssn of the EMPLOYEE tuple with Ssn = ‘999887777’ to ‘987654321’.

Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn.