FIT3171 Databases

Week 7 Tutorial Activities

Creating and Populating the Database

FIT Database Teaching Team

Complete the week 7 activities:

FIT3171 2022 S1

FIT3171 Databases

Author: FIT Database Teaching Team

Important

Remember before starting any lab activity which involves working with files, first use SQL Developer to pull from the FIT GitLab server so as to ensure your local files and the FIT Git Lab server files are in sync. *During this activity, you will be creating a set of sql scripts, these need to be sent to the FIT GitLab server.*

Learning Objectives:

- Create tables in a database
- Add new records to a table
- Use ORACLE's sequences to generate keys
- Remove tables from a database

Important Note

The concepts and syntax we cover in this week can be used to check the validity of the schema file that you generate from Data Modeler for assignment 1B. However please note that you **MUST NOT** edit the schema file generated by Data Modeller in any manner other than adding the heading and turn off of echo/spool at the bottom of the file - the actual **generated SQL code must not be changed in any way**. If you believe the SQL generated is incorrect, for assignment 1B, you need to go back and correct the issue in your logical model and then regenerate the schema.

7.1. SQL Data Definition Language (DDL) - Class Discussion

When creating schema files, you should always also create a drop file or add the drop commands to the top of your schema file. You should drop the tables using the

```
drop table tablename purge;
```

syntax. If you use this syntax, the drop table statements should list tables in the reverse order of your create table order so that FK relationships will be able to be removed successfully.

You could also use:

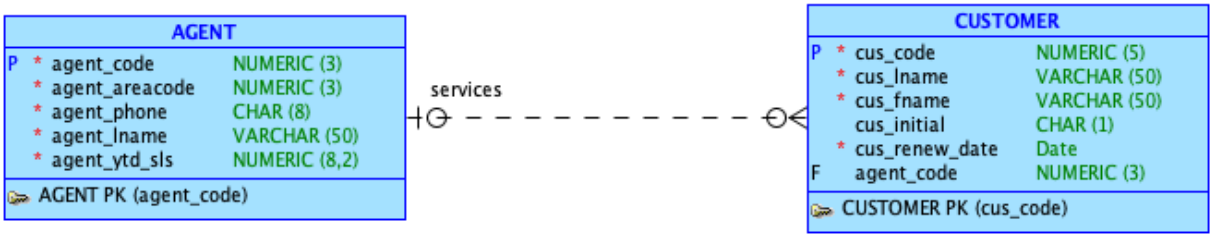
```
drop table tablename cascade constraints purge;
```

syntax to drop the table and all constraints belonging to the table. If you use this syntax, the order of table deletion does not matter.

Should a syntax error occur while testing your schema, you simply need to run the drop commands to remove any tables which may have been created.

An excellent summary of the Oracle data types and version restrictions is available from: https://www.oracletutorial.com/oracle-basics/oracle-data-types/

For this unit, we make use of **CHAR, VARCHAR2, NUMBER** and **DATE ONLY**



The data model above represents figure 3.3 from Coronel & Morris. Note that this diagram is not a logical model. ***Logical models do not show data types and data sizes*.** Also note this is not an Oracle physical model, since the data types are not standard Oracle types - NUMERIC should be NUMBER and VARCHAR should be VARCHAR2.

There are two different ways of coding this model as a set of create table statements as discussed in the following subsections.

7.1.1 Using table constraints

SQL constraints are classified as column or table constraints; depending on which item they are attached to:

```
CREATE TABLE agent (
    agent_code      NUMBER(3) CONSTRAINT agent_pk PRIMARY KEY,
    agent_areacode  NUMBER(3) NOT NULL,
    agent_phone     CHAR(8) NOT NULL,
    agent_lname     VARCHAR2(50) NOT NULL,
    agent_ytd_sls   NUMBER(8, 2) NOT NULL
);
```

This is a declaration of the primary key as a column constraint

```
CREATE TABLE agent (
    agent_code      NUMBER(3) NOT NULL,
    agent_areacode  NUMBER(3) NOT NULL,
    agent_phone     CHAR(8) NOT NULL,
    agent_lname     VARCHAR2(50) NOT NULL,
    agent_ytd_sls   NUMBER(8, 2) NOT NULL,
    CONSTRAINT agent_pk PRIMARY KEY ( agent_code )
);
```

Here the primary key has been declared as a table constraint, at the end of the table after all column declarations have been completed. In some circumstances, for example, a composite primary key you must use a table constraint since a column constraint can only refer to a single column.

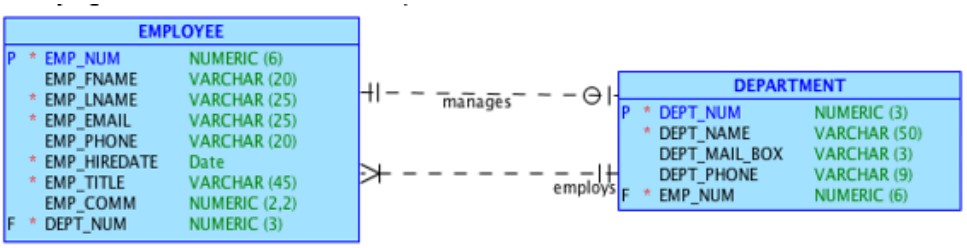
The create table statements for the two tables in fig 3-3 would be:

```
CREATE TABLE agent (
    agent_code      NUMBER(3) NOT NULL,
    agent_areacode  NUMBER(3) NOT NULL,
    agent_phone     CHAR(8) NOT NULL,
    agent_lname     VARCHAR2(50) NOT NULL,
    agent_ytd_sls   NUMBER(8, 2) NOT NULL,
    CONSTRAINT agent_pk PRIMARY KEY ( agent_code )
);

COMMENT ON COLUMN agent.agent_code IS
    'agent code (unique for each agent)';

COMMENT ON COLUMN agent.agent_areacode IS
    'area code of agent';

COMMENT ON COLUMN agent.agent_phone IS
    'phone number of agent code';

COMMENT ON COLUMN agent.agent_lname IS
    'last name of agent';

COMMENT ON COLUMN agent.agent_ytd_sls IS
    'year to date sales made by agent';

CREATE TABLE customer (
    cus_code        NUMBER(5) NOT NULL,
    cus_lname       VARCHAR2(50) NOT NULL,
    cus_fname       VARCHAR2(50) NOT NULL,
    cus_initial     CHAR(1),
    cus_renew_date  DATE NOT NULL,
    agent_code      NUMBER(3),
    CONSTRAINT customer_pk PRIMARY KEY ( cus_code ),
    CONSTRAINT customer_agent_fk FOREIGN KEY ( agent_code )
        REFERENCES agent ( agent_code )
            ON DELETE SET NULL
);

COMMENT ON COLUMN customer.cus_code IS
    'customer code (unique for each customer)';

COMMENT ON COLUMN customer.cus_lname IS
    'last name of customer';

COMMENT ON COLUMN customer.cus_fname IS
    'first name of customer';

COMMENT ON COLUMN customer.cus_initial IS
    'initial of customer (not mandatory)';

COMMENT ON COLUMN customer.cus_renew_date IS
    'insurance renewal date of customer';

COMMENT ON COLUMN customer.agent_code IS
    'agent code (unique for each agent)';
```

The inclusion of the referential integrity rule *on delete set null* in the above create table statement is **appropriate in this scenario** - when an agent leaves, a reasonable approach would be to set the foreign key for that agent's customers to null. The default on delete restrict (which you do not specify, simply omit an on delete clause) would also be an alternative approach. Using *on delete cascade* would not be appropriate since this would cause the customers of the agent who left to also be deleted. When coding a foreign key definition you **must always consider what is a suitable on delete approach (RESTRICT, CASCADE, NULLIFY) for the scenario you are working with**.

7.1.2 Using ALTER table commands

In some circumstances, this approach of defining the foreign keys as part of the table definitions cannot be used. Observe the data model below. Note that this diagram is not a logical model. Logical model does not show data types and data sizes. Can you see what the issue is with trying to create the two tables depicted below?



In such a situation an alternative approach to declaring constraints needs to be adopted.

In this approach, the tables are declared without constraints and then the constraints are applied via the ALTER TABLE command (see section 7.5 of Coronel & Morris).

```
CREATE TABLE agent (
    agent_code      NUMBER(3) NOT NULL,
    agent_areacode  NUMBER(3) NOT NULL,
    agent_phone     CHAR(8) NOT NULL,
    agent_lname     VARCHAR2(50) NOT NULL,
    agent_ytd_sls   NUMBER(8, 2) NOT NULL
);

COMMENT ON COLUMN agent.agent_code IS
    'agent code (unique for each agent)';

COMMENT ON COLUMN agent.agent_areacode IS
    'area code of agent';

COMMENT ON COLUMN agent.agent_phone IS
    'agent phone number';

COMMENT ON COLUMN agent.agent_lname IS
    'agent last name';

COMMENT ON COLUMN agent.agent_ytd_sls IS
    'year to date sales made by agent';

ALTER TABLE agent ADD CONSTRAINT agent_pk PRIMARY KEY ( agent_code );
```

```
CREATE TABLE customer (
    cus_code        NUMBER(5) NOT NULL,
    cus_lname       VARCHAR2(50) NOT NULL,
    cus_fname       VARCHAR2(50) NOT NULL,
    cus_initial     CHAR(1),
    cus_renew_date  DATE NOT NULL,
    agent_code      NUMBER(3)
);

COMMENT ON COLUMN customer.cus_code IS
    'customer code (unique for each customer)';

COMMENT ON COLUMN customer.cus_lname IS
    'customer last name';

COMMENT ON COLUMN customer.cus_fname IS
    'customer first name';

COMMENT ON COLUMN customer.cus_initial IS
    'customer initial (not mandatory)';

COMMENT ON COLUMN customer.cus_renew_date IS
    'customer insurance renewal date';

COMMENT ON COLUMN customer.agent_code IS
    'agent code (unique for each agent)';

ALTER TABLE customer ADD CONSTRAINT customer_pk PRIMARY KEY ( cus_code );

ALTER TABLE customer
    ADD CONSTRAINT customer_agent_fk FOREIGN KEY ( agent_code )
        REFERENCES agent ( agent_code )
            ON DELETE SET NULL;
```

Remember, from above, when coding a foreign key definition you **must always consider what is a suitable on delete approach (RESTRICT, CASCADE, NULLIFY) for the scenario you are working with**.

Using an ALTER Table approach **is the best method** since it cannot fail due to missing required tables for foreign key constraints. Using this approach the tables can be created in alphabetical order and then the required constraints applied. This is the approach used by SQL Developer (and most commercial software) when creating schema files.

After creating the tables we need to insert the data, for AGENT the insert will have the form:

```
insert into agent values (501,713,'228-1249','Alby',132735.75);
```

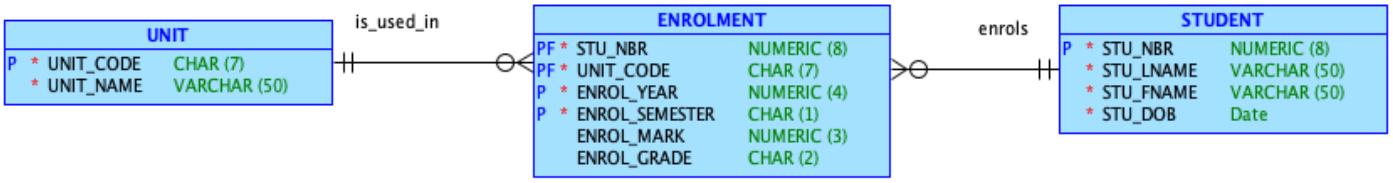
for customer:

```
insert into customer values(10010,'Ramas','Alfred','A',
    to_date('05-Apr-2019','dd-Mon-yyyy'),501);
```

To insert a date, you must use the to_date function which converts your string input to a date.

7.2 CREATing tables

Using the data model from the week 6 workshop for student, unit and enrolment shown below, please complete the following tasks.



TASKS

1. Download the schema start file (week7_schema_alter.sql) from Moodle and save this **week7_schema_alter.sql** under the App07 folder in your local repository
2. Using this file complete the schema to create these three tables, noting the following ***extra*** constraints:
   a. stu_nbr > 10000000
   b. unit_name is unique in the UNIT table
   c. enrol_semester can only contain the value of 1 or 2 or 3.
3. In implementing these constraints you will need to make use of CHECK clauses (see Coronel & Morris section 7.2.6 or here: https://www.oracletutorial.com/oracle-basics/oracle-check-constraint/).
4. Ensure your script file has appropriate comments in the header, includes the required drop commands and includes echo on and echo off commands.
5. Run your script and create the three required tables.
6. Save the output from this run. To save the output we make use of the inbuilt Oracle SPOOL command, as you did with your data Modeller generated schemas. To include the result of the SQL commands in the output we make use of the inbuilt Oracle SET ECHO command. To use SPOOL and SET ECHO, place as the top line in your schema file:

```
set echo on
spool week7_schema_alter_output.txt
```

and as the last line in your script file

```
spool off
set echo off
```

This will produce a file, in the same folder that your script is saved in, called week7_schema__alter_output.txt which contains the full run of your SQL script. Note that the filename you are spooling to must not contain spaces or special characters (use letters, numbers and _ only)

7.3. INSERTing data into the database

Create a new sql file: File → New → Database Files → SQL File, and name the file as **week7_insert.sql** and save this file under App07 in your local repository. Write the required SQL statements for section 7.3.1 to 7.3.4 in this file.

Save the output from this run as **week7_insert_output.txt**. To save the output, use of the inbuilt Oracle SPOOL command as discussed in section 7.2.

7.3.1 Basic INSERT statement

In this exercise, you will enter the data into the database using INSERT statements with the following assumptions:

- the database currently does not have any existing data, and
- the primary key is *not generated automatically* by the DBMS.

TASKS

1. Write SQL INSERT statements within the file to add the following data into the specified tables
2. A file is available on Moodle (studentdata.txt) which has this data as a starting point to build the required insert statements. Download the file from Moodle and open studentdata.txt in SQL Developer, then copy and paste the contents to build your insert statements. In building the insert statements:
    a. numeric data (ie. numbers) must not be enclosed in quotation marks,
    b. character data must be enclosed within single quotes, and
    c. date data must make use of to_date to convert strings to dates
3. Ensure you make use of COMMIT to make your changes permanent. At this early stage you should add a commit, as a minimum, at the end of each of the tasks you complete for this tutorial. Remember commit is only used for Insert, Update and Delete ie DML commands. In the following weeks we will return and examine transactions in more detail and take a more granular approach to establishing transaction boundaries
4. Check that your data has inserted correctly by using the SQL command SELECT * FROM *tablename* **and** by using the SQL GUI (select the table in the right-hand list and then select the Data tab).

STUDENT

stu_nbr	stu_lname	stu_fname	stu_dob
11111111	Bloggs	Fred	01-Jan-2003
11111112	Nice	Nick	10-Oct-2004
11111113	Wheat	Wendy	05-May-2005
11111114	Sheen	Cindy	25-Dec-2004

UNIT

unit_code	unit_name
FIT9999	FIT Last Unit
FIT9132	Introduction to Databases
FIT9161	Project
FIT5111	Student's Life

ENROLMENT

stu_nbr	unit_code	enrol_year	enrol_semester	enrol_mark	enrol_grade
11111111	FIT9132	2020	1	35	N
11111111	FIT9161	2020	1	61	C
11111111	FIT9132	2020	2	42	N
11111111	FIT5111	2020	2	76	D
11111111	FIT9132	2021	1		
11111112	FIT9132	2020	2	83	HD
11111112	FIT9161	2020	2	79	D
11111113	FIT9132	2021	1		
11111113	FIT5111	2021	1		
11111114	FIT9132	2021	1		
11111114	FIT5111	2021	1		

7.3.2 Using SEQUENCEs in an INSERT statement

In the previous exercises, you have entered the primary key value manually in the INSERT statements. In the situation where a SEQUENCE is available, you should use the sequence mechanism to generate the value of the primary key.

TASKS

1. Create a sequence for the STUDENT table called STUDENT_SEQ
    - Create a sequence for the STUDENT table called STUDENT_SEQ that starts at 11111115 and increases by 1.
    - Check that the sequence exists in two ways (using SQL and browsing your SQL Developer connection objects).
2. Add a new student (MICKEY MOUSE)
    - Use the student sequence - pick any STU_DOB you wish.
    - Check that your insert worked.
    - Add an enrollment for this student to the unit FIT9132 in semester 1 2021.

7.3.3 Advanced INSERT

We have learned how to add data into the database in the previous exercises through the use of INSERT statements. In those exercises, the INSERT statements were created as a single script assuming that the data is all added at the same time, such as at the beginning when the tables are created. On some occasions, new data is added after some data already exists in the database. In this situation, it is a good idea to use a combination of INSERT and SELECT statements.

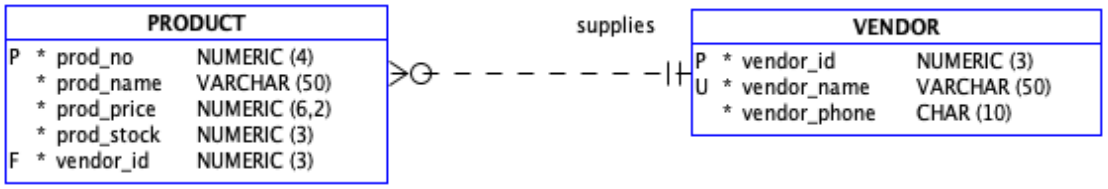
A SELECT statement is an SQL statement that we use to retrieve data from a database. An example of a SELECT statement would be:

```
SELECT vendor_id
FROM vendor
WHERE vendor_name = 'Seagate';
```

The above SQL statement consists of three SQL clauses SELECT, FROM and WHERE. The SELECT clause is used to declare which column(s) are to be displayed in the output. The FROM clause is used to declare from which table the data needs to be retrieved. The WHERE clause is used to declare which rows are to be retrieved. In the above SQL select, any row that has the vendor_name equal to 'Seagate' will be retrieved. The SQL SELECT statement will be covered in more detail in the future module, retrieving data from the database.

For our exercise on using the advanced INSERT statement, consider the following model depicting VENDOR and PRODUCT.

Assume we want to add vendors and the products they supply into a set of tables represented by:



A suitable schema would be:

```
DROP TABLE product PURGE;

DROP TABLE vendor PURGE;

DROP SEQUENCE prod_no_seq;

DROP SEQUENCE vendor_id_seq;

CREATE TABLE product (
    prod_no     NUMBER(4) NOT NULL,
    prod_name   VARCHAR2(50) NOT NULL,
    prod_price  NUMBER(6, 2) NOT NULL,
    prod_stock  NUMBER(3) NOT NULL,
    vendor_id   NUMBER(3) NOT NULL
);

COMMENT ON COLUMN product.prod_no IS
    'product number (unique for each product)';
```

```
COMMENT ON COLUMN product.prod_name IS
    'product name';

COMMENT ON COLUMN product.prod_price IS
    'product price';

COMMENT ON COLUMN product.prod_stock IS
    'product on hold (stock)';

COMMENT ON COLUMN product.vendor_id IS
    'vendor id (unique for each vendor)';

ALTER TABLE product ADD CONSTRAINT product_pk PRIMARY KEY ( prod_no );

CREATE TABLE vendor (
    vendor_id     NUMBER(3) NOT NULL,
    vendor_name   VARCHAR2(50) NOT NULL,
    vendor_phone  CHAR(10) NOT NULL
);

COMMENT ON COLUMN vendor.vendor_id IS
    'vendor id (unique for each vendor)';

COMMENT ON COLUMN vendor.vendor_name IS
    'vendor name';

COMMENT ON COLUMN vendor.vendor_phone IS
    'vendor phone';

ALTER TABLE vendor ADD CONSTRAINT vendor_pk PRIMARY KEY ( vendor_id );

ALTER TABLE vendor ADD CONSTRAINT vendor_un UNIQUE ( vendor_name );

ALTER TABLE product
    ADD CONSTRAINT vendor_product_fk FOREIGN KEY ( vendor_id )
        REFERENCES vendor ( vendor_id )
            ON DELETE CASCADE;

CREATE SEQUENCE prod_no_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE vendor_id_seq START WITH 1 INCREMENT BY 1;
```

There are two ways in which we can perform the INSERT.

1. Use the nextval and currval of the sequences.

```
-- Add Vendor 1 and the products they supply
 insert into vendor values (vendor_id_SEQ.nextval,
        'Western Digital', '1234567890');
 insert into product values (prod_no_SEQ.nextval,
        '2TB My Cloud Drive',195,5,vendor_id_SEQ.currval);
 insert into product values (prod_no_SEQ.nextval,
        '1TB Portable Hard Drive',76,4,vendor_id_SEQ.currval);
 insert into product values (prod_no_SEQ.nextval,
        'Live Media Player',119,2,vendor_id_SEQ.currval);

 commit;
```

```
-- Add Vendor 2 and the products they supply
 insert into vendor values (vendor_id_SEQ.nextval,'Seagate',
        '2468101234');
 insert into product values (prod_no_SEQ.nextval,
        '2TB Desktop Drive',94,12,vendor_id_SEQ.currval);
 insert into product values (prod_no_SEQ.nextval,
        '4TB 4 Bay NAS',76,4,vendor_id_SEQ.currval);
 insert into product values (prod_no_SEQ.nextval,
        '2TB Central Personal Storage' ,169,5, vendor_id_SEQ.currval);
 commit;
```

2. Use the nextval in combination with the SELECT statement.
- Add a new product for a vendor at a subsequent time (vendor names will be unique - note the U in the model above and the vendor_un constraint in the schema)

```
insert into product values (prod_no_SEQ.nextval,
        'GoFlex Thunderbolt Adaptor',134,2,
        (select vendor_id from vendor where vendor_name = 'Seagate'));
commit;
```

In subsequent weeks you will see that the same concept can be used with other data manipulation statements such as UPDATE and DELETE.

TASKS

1. A new student has started a course and needs to enrol into "Introduction to databases". Enter the new student's details and his/her enrollment to the database using the sequence in combination with a SELECT statement. You can make up details of the new student and when they will attempt "Introduction to databases".
    - You must not do a manual lookup to find the unit code of the "Introduction to Databases".

7.3.4 Creating a table and inserting data as a single SQL statement.

A table can also be created based on an existing table, and immediately populated with contents by using a SELECT statement within the CREATE TABLE statement.

For example, to create a table called FIT9132_STUDENT which contains the enrolment details of all students who have been or are currently enrolled in FIT9132, we would use:

```
create table FIT9132_STUDENT
as select *
from enrolment
where unit_code = 'FIT9132';
```

Here, we use the SELECT statement to retrieve all columns (the wildcard "*" represents all columns) from the table enrollment, but only those rows with a value of the unit_code equal to FIT9132.

TASKS

1. Create a table called FIT5111_STUDENT. The table should contain all enrolments for the unit FIT5111.
2. Check the table exists.
3. List the contents of the table.

7.4. ALTERing the structure of a database table

TASKS

1. Create a new sql file: File → New → Database Files → SQL File, and name the file as **week7_altertable.sql** and save this file under Tut07 in your local repository
2. Add a new column to the UNIT table which will represent credit points for the unit (hint use the ALTER command). The default value for the new column should be 6 points.
3. Insert a new unit after you have added the new column. You can make up the details of the new unit.
4. Check that the new insert has worked correctly.

Important

You need to get into the habit of establishing this as a standard FIT3171 workflow - Pull at the start of your working session, work on the activities you wish to/are able to complete during this session, add all(stage), commit changes and then push the changes back to the FIT GitLab server