**First class, general info of the class**

(Notes #1a)

**Topics**

1. SQL\* Developer and the sample tables
2. Some SQL concepts, commands we may need in this quarter.

This note is special for this spring quarter. This class will use the COL recording from last winter quarter, I write down these, hope that will help you in your study and homework.

**Prerequisites** for this course.

This class is about programming, it needs your commitment of time and efforts to learn, review and complete the homework assignments. That is the learning curve, there is no shortcut.

This class assumes that the students have finished certain prerequisite courses, and the students have some background in relational database, SQL language and programming language.

Homework #1 and #2 can be used as self-evaluation.

This class will concentrate on PL/SQL language. Inevitably, we need some knowledge related to Database. PL/SQL is kind of Oracle version 3GL, dedicated to Oracle database environment. Technically it has many same/similar features, structures and concepts as other 3GL languages. Certain repetitions, comparisons may strengthen our ability both in DB and general programming.

The file note1c\_Review\_SQL is my former notes when teaching SQL, just for your reference. It is not required to go through in details.

Here, we will go over some basic concepts we might need in this course.

1. **SQL\* Developer** and the sample tables from Oracle.

If you have no experience with SQL \*developer in former class(es), please find out if you like it or not as soon as possible. We will work on SQL\*Developer environment all the quarter, also you will need it to complete your midterm and final exams.

Please check the folder of “Scripts for creating sample tables” under “Content” in D2L. There are two files for creating Oracle Emp, Employees tables. These are not only useful for this class, but also help you to read Oracle SQL and PL/SQL manuals in the future.

There are 7 tables in Oracle HR sample.

Employees (EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, HIRE\_DATE, JOB\_ID, SALARY, COMMISSION\_PCT, MANAGER\_ID, DEPARTMENT\_ID),

Departments,

Countries, Regions, Jobs, Job\_History, Locations.

We will use the Employees table, and a few times, also use Departments table (for joining)

CREATE TABLE DEPARTMENTS (

DEPARTMENT\_ID NUMBER (4) NOT NULL,

DEPARTMENT\_NAME VARCHAR2 (30 BYTE) NOT NULL,

MANAGER\_ID NUMBER (6),

LOCATION\_ID NUMBER (4)

) ;

CREATE TABLE EMPLOYEES (

EMPLOYEE\_ID NUMBER (6) NOT NULL,

FIRST\_NAME VARCHAR2 (20 BYTE),

LAST\_NAME VARCHAR2 (25 BYTE) NOT NULL,

EMAIL VARCHAR2 (25 BYTE) NOT NULL,

PHONE\_NUMBER VARCHAR2 (20 BYTE),

HIRE\_DATE DATE NOT NULL,

JOB\_ID VARCHAR2 (10 BYTE) NOT NULL,

SALARY NUMBER (8,2),

COMMISSION\_PCT NUMBER (2,2),

MANAGER\_ID NUMBER (6),

DEPARTMENT\_ID NUMBER (4)

) ;

Oracle also provides a set of tables (simpler):

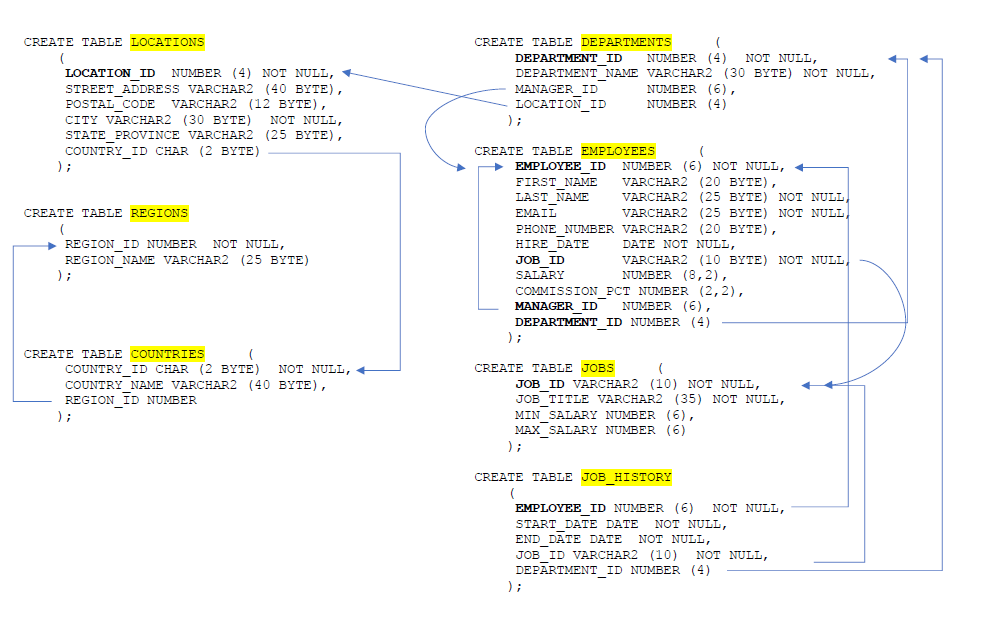
EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO);

Dept (DEPTNO, DNAME, LOC);

Bonus;

Salgrade.

In this class, we will use both Emp/Dept and Employees/Departments tables.



A quick review of DB design:

Tables in a business connected each other via FK-PK (foreign key to primary key or unique key).

In the relation database, according to normalization, business information are divided into small parts and created as tables. Each table is kind independent (functional dependency). One table at least will share one (or a set of columns) column with another table, and usually they have a relationship as primary (unique) key and foreign key.

This serves the purpose to keep the data integrity. But when retrieving data, SQL statement has to get pieces of information from different tables. Hence joining several tables are needed (such as tableA.pk = tableB.fk), plus subqueries, where conditions, etc. These are the tricks in SQL commands.

The above diagram, I marked the referential integrity relationship (Pk-FK) for the tables in HR sample.

If the requirements of information are not that critical accurate for every moment, materialized view will do the work. It is kind of denormalization, good for queries.

Note, in table employees, there is a relationship between the manager\_id and employee\_id (similar ‘mgr’ with ‘empno’ in emp table). Managers must be employees, the manager\_id is a foreign key to the employee\_id in the same table of employees.

The department\_id in Employees table is the foreign key for department\_id in the Departments table.

**2. A quick review for some SQL commands** (that we may need in this quarter)

A sample **DDL syntax** such as:

CREATE TABLE table\_name

(column\_name datatype [default expr] [column\_constraint],

column\_name datatype [default expr] [column\_constraint],

...

primary\_key\_definition,

foreign\_key\_definition, foreign\_key\_definition);

Follow the “CREATE” key word is the object type, such as table, user, sequence, etc.

Later we will see similar syntax:

CREATE **[OR REPLACE]** PACKAGE pkg\_name

Many of the CREATE commands in this class allowed “CREATE **[OR REPLACE]”.**

Notice, most of the DDL commands do not have the option of [OR REPLACE].

(except CREATE [OR REPLACE] VIEW view\_name as select ...)

All these objects built by the “create” commands will be stored in database, and accessible for the owner (and the grantees). These objects are part of the schema (under that owner/user name). Later in this class, many times, you will see the terminologies of schema, package and PL/SQL block. for details, refer to appendix in note2\_PLSQL\_fundamentals.

The command below is useful to build a backup table. Because after some DML commands, your table contents could be quite different from when just created.

Create table with Subquery

# CREATE TABLE x AS SELECT \* FROM y;

This command will create a new table x which is a duplication of table y including all the records and the “not null” (only) constraints. For this course, we will not emphasize the constraints.

Student can create their tables by simply copying from the instructor’s tables. After the instructor created his/her tables, the instructor runs the “GRANT” commands, for example:

GRANT SELECT ON employees to public;

This will allow every user in this Oracle instance to “read” the table.

From student side, type the command:

CREATE TABLE employees AS SELECT \* FROM czhang.employees ;

This command will create a table “employee” owned by student account.

(Note, it will not copy the constraints except “Not Null”).

Function **SYSDATE**  returns the current date and the time. (we will use this function a lot)

SELECT sysdate FROM DUAL;

-- DUAL is a dummy table in Oracle. (Dual has one column and one tuple)

**COMMIT** [WORK]:

To end your current transaction and make permanent all changes performed

in the transaction.

**ROLLBACK** [WORK]:

To undo work done in the current transaction (up to last commit).

Computer architecture: CPU < -- > RAM < -- > Database stored in the hard disk.

RAM is a volatile memory storage, all the info there will be gone once the power is off.

Hard disk is the permanent storage for data – where DB resides.

All the changes to the contents of tables by your DML commands (e.g. insert, update and delete) are temporarily stored in RAM, these can be cancelled by “rollback”, or save to permanent storage by explicit “commit” command.

Implicit commit: in the following cases, the system will automatically commit the transaction for you.

* Whenever Oracle executes a DDL (such as “create”, “drop” command)
* “Autocommit” is set on in the environment
* A normal exit from Oracle (such as type exit in the SQL\* Developer command line; an abnormal exit – such as click the “exit” in the window box, will cause rollback ).

GRANT privilege [, privilege ...] on table\_name TO user [,user ...] ;

If you want to grant the read (select) on some table for everyone/someone:

GRANT SELECT ON EMPLOYEES TO public; -- Or

GRANT SELECT, UPDATE ON EMPLOYEES TO usera, userb;

To cancel it:

REVOKE SELECT ON EMPLOYEES FROM public;

You can put multiple privileges/multiple users in a single statement;

but you can ONLY put a single table (or view) in one statement.

Note, as course work, most times, when we work on SQL\*developer, we are working by oneself alone. That means, student A logs in to his/her own account, all the tables, views, functions, types are created by student A. In business environment, say, you work as a programmer in a company, (unless you are DBA) you will use the tables created by others. They granted the privilege of accessing to these tables for you. Others may have created some functions, packages for the business, they may offer you the access. In these cases, you may need to “qualify” these tables, functions, packages (put the owner user name before the table name, such as czhang.employees). For these and other reasons, later in this class, we will discuss the concepts of packages. Even before we formally discuss the package in the last weeks, the course work will mention about the “package” many times.

Join a table to itself.

Select statement below will list the employee name with their boss name.

SELECT e.ename as emp, b.ename as boss

FROM emp e, emp b

WHERE b.empno = e.mgr;

Some Functions we may need for this course.

UPPER(n) - Converts character string to uppercase

SELECT empno FROM emp

WHERE upper (name)= 'KING'; -- we do not know the rules for names

FLOOR(n) returns largest integer equal to or less than n

SELECT ename, floor (months\_between (sysdate, HireDate)/12) Year\_worked

from emp

WHERE deptno = 10;

Sysdate: returns the current date (and time), requires no arguments.

Months\_between (d1, d2) returns number of months between date d1 and d2.

Select sysdate "TODAY" from Dual;

Some character-manipulation Functions you will need for your future homework,

SUBSTR (char, position, string\_length)

-- it returns a portion of char, beginning at char “position”, as string\_length long.

RPAD (expr 1, n [, expr2])

-- it reads expr1, right-padded to length n characters with expr2 replicated as many times as necessary. The default of expr2 is a single blank (' ').

Function sample Result

SUBSTR ('HelloWorld', 1, 5) Hello -- returns a string of a determined length

RPAD (ename,10, ' ') KING

-- ename is table column name here, right-pads space to length 10

RPAD (ename,10) KING

CONCAT ('Hello', 'World') HelloWorld -- joins literals together

CONCAT (char1, char2) -- It returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes char, chvarchar2. This function is equivalent to the concatenation operator (||).

Concatenation operator ||

It appends one string operand to another.

-- most times in this class, we will use this '||', instead of “concat”.

Also, we may repeat this concatenation operator.

SELECT last\_name || ', ' || first\_name AS Full\_Name

FROM employees

WHERE employee\_id = 152;

Result:

FULL\_NAME

----------------

Hall, Peter

Conversion Functions:

TO\_CHAR (x) - converts x value to VARCHAR2, details refer to page 9 in note7b\_functions.

SELECT TO\_CHAR (1234.89, '$9,999.00') FROM dual; -- RESULT: $1,234.89

TO\_CHAR(sysdate, 'Month DD, YYYY') -- RESULT: March 26, 2020

**COUNT** (**{** \* |[ Distinct | All ] expr **}**)

Returns the number of rows in the query.

SELECT count (\*) number\_of\_emp

FROM emp

Where deptno = 10;

**Handling NULLs:**

NULL means information missing or unknown.

NULL is different from Blank space which is a character and its ASCII hexadecimal value

is 20. Null is also different from zero which is number.

If a row lacks value for a particular column, that column is said to be NULL.

SELECT first\_name ||' '||last\_name "full Name"

FROM employees

WHERE department\_id ***IS*** NULL;

-- never use "= NULL" in the "where" clause.

NVL (expr1, expr2) function: converts a null (expr1) to an actual value

-- expr1: the source that may contain null, returns expr1 if it is not null.

-- expr2: if expr1 is null, then function returns expr2.

SELECT empno , ename , (***sal + comm) total\_income*** FROM emp;

-- This expression ( ***sal + comm*** )will not give correct result when comm is null.

because (anything + null) results in null.

The correct statement when related to possible NULL should be as below:

SELECT empno, ename, sal + NVL (comm, 0) total\_income

FROM emp;

-- if “comm” value is null, then convert it to zero.

Similar query related to Employees table:

COLUMN full\_name format A25

SELECT employee\_id, first\_name || ' '||last\_name full\_name,

salary \* (1+ NVL(commission\_PCT, 0)) total\_income

FROM employees

WHERE department\_id = 20;