SQQL

Assignment 2   
Database Management -   
 Hands-on   
Spring 2018)

***laverty@rmu.edu***

Contents

[**1.0 Reviewing the Project Management Scripts 2**](#_Toc482356910)

[**1.1 create\_project.sql Script 2**](#_Toc482356911)

[**1.2 insert\_project. sql Script 2**](#_Toc482356912)

[**1.3 create\_view\_project.sql Script 2**](#_Toc482356913)

[**1.4 Sample Data and Table Structure 2**](#_Toc482356914)

[**1.5 Relationship between DEPT and EMP 2**](#_Toc482356915)

[**1.6 Relationship between DEPT and PROJ 2**](#_Toc482356916)

[**1.7 Relationships between EMP and PROJ with EMPPROJACT 2**](#_Toc482356917)

[**2.0 Questions - Database Project Design 2**](#_Toc482356918)

[**2.1 Questions - Primary Key Constraint 2**](#_Toc482356919)

[**2.2 Questions - Entity and Referential Integrity 2**](#_Toc482356920)

[**2.11 Questions - Check Constraints 2**](#_Toc482356921)

[**Requirement 3. Simple SQL Select Queries 2**](#_Toc482356922)

[**3.1 Basic Select Statement 2**](#_Toc482356923)

[**3.2 Order By Clause 2**](#_Toc482356924)

[**3.3 DISTINCT Clause 2**](#_Toc482356925)

[**3.4 Your Requirement Example 2**](#_Toc482356926)

[**3.5 WHERE Clause 2**](#_Toc482356927)

[**3.6 Comparison Operators 2**](#_Toc482356928)

[**3.7 Logical Operators – AND, OR, NOT 2**](#_Toc482356929)

[**3.8 IN Operator 2**](#_Toc482356930)

[**3.9 BETWEEN Operator 2**](#_Toc482356931)

[**3.10 NULL Comparison 2**](#_Toc482356932)

[**3.11 LIKE Operator – Partial String Match 2**](#_Toc482356933)

[**3.12 Questions - SELECT review 2**](#_Toc482356934)

[**Requirement 4. SQL Scalar Functions and Arithmetic 2**](#_Toc482356935)

[**4.1 Calculated Columns 2**](#_Toc482356936)

[**4.2 Naming Calculated Columns - AS 2**](#_Toc482356937)

[**4.3 COALESCE Function 2**](#_Toc482356938)

[**4.4 Calculate Values with the WHERE Clause 2**](#_Toc482356939)

[**4.5 SUBSTR Function 2**](#_Toc482356940)

[**4.6 Concatenation 2**](#_Toc482356941)

[**Requirement 5. Column Functions and Grouping 2**](#_Toc482356942)

[**5.1 Basic Column Functions 2**](#_Toc482356943)

[**5.2 GROUP BY 2**](#_Toc482356944)

[**5.3 Incorrect GROUP BY Statements 2**](#_Toc482356945)

[**5.4 GROUP BY, ORDER BY 2**](#_Toc482356946)

[**5.5 GROUP BY, WHERE, and HAVING 2**](#_Toc482356947)

[**5.6 Another GROUP BY and Having Example 2**](#_Toc482356948)

**Enter your Name Here 🡺**

**It is recommended that you use Table of Contents at the beginning of the tutorial to review and navigate to the concept presented in the review question. Students will find that using the document FIND tool or searching GOOGLE may also be valuable for researching the review question answer.**

# 1.0 Reviewing the Project Management Scripts

## 1.1 create\_project.sql Script

The previous requirements required you to execute several scripts that would create four tables: EMP, DEPT, PROJ ,EMPPROJACT, The SQL DDL Syntax of this script is provided below.

**CREATE TABLE EMP (**

**EMPNO CHAR(6) NOT NULL ,**

**FIRSTNME VARCHAR(12) NOT NULL ,**

**MIDINIT CHAR(1) NOT NULL ,**

**LASTNAME VARCHAR(15) NOT NULL ,**

**WORKDEPT CHAR(3) ,**

**PHONENO CHAR(4) ,**

**HIREDATE DATE ,**

**JOB CHAR(8) ,**

**EDLEVEL SMALLINT NOT NULL ,**

**SEX CHAR(1) ,**

**BIRTHDATE DATE ,**

**SALARY DECIMAL(9,2) ,**

**BONUS DECIMAL(9,2) ,**

**COMM DECIMAL(9,2) ) ;**

**CREATE TABLE DEPT (**

**DEPTNO CHAR(3) NOT NULL ,**

**DEPTNAME VARCHAR(29) NOT NULL ,**

**MGRNO CHAR(6) ,**

**ADMRDEPT CHAR(3) NOT NULL ,**

**LOCATION CHAR(16) ) ;**

**CREATE TABLE PROJ (**

**PROJNO CHAR(6) NOT NULL ,**

**PROJNAME VARCHAR(24) NOT NULL ,**

**DEPTNO CHAR(3) NOT NULL ,**

**RESPEMP CHAR(6) NOT NULL ,**

**PRSTAFF DECIMAL(5,2) ,**

**PRSTDATE DATE ,**

**PRENDATE DATE ,**

**MAJPROJ CHAR(6) ) ;**

**CREATE TABLE EMPPROJACT (**

**EMPNO CHAR(6) NOT NULL ,**

**PROJNO CHAR(6) NOT NULL ,**

**ACTNO SMALLINT NOT NULL ,**

**EMPTIME DECIMAL(5,2) ,**

**EMSTDATE DATE ,**

**EMENDATE DATE ) ;**

**COMMIT;**

Unlike storing data in an Excel Workbook, you must plan the organization of the data to be stored and the type of data to be stored. In Excel, you have the option of labeling a column with a name, e.g., EMPNO, In Excel a column name is not required nor are there any Excel rules or naming a column name. But a database table requires a valid column name which must follow the syntax rules of the database that you are using. For example, DB2 and Oracle's column names are limited to 30 characters, must start with a letter, should avoid most special characters, and by default is not case sensitive.

When you type data under an Excel column name guesses the data type of data enter. A data type as used in databases and all programming languages limits the type of data that may be stored. For example, a numeric data type permits only numeric digits, no alphabetic or special characters. A numeric data type may be used in mathematical calculations. Excel describes the alphanumeric data types as a "label", but databases use the term char or varchar. Labels, char or varchar datatypes cannot be used in mathematical calculations and the data must be enclosed in quotes

When you use design a database, you must first design the organization of the data as tables and relationships to tables. Then you must define the names of columns or attributes which stores data in a table. And then you must declare a data type for each column which will limit the type of data entered into a column and the operations that may be performed on the column.

In short, you must plan for structure, data, and data types to be stored BEFORE you store or use the data. This is the purpose of the SQL CREATE TABLE DDL (data definition language) statement.

## 1.2 insert\_project. sql Script

**DELETE FROM EMP;**

**-- INSERT NEW ROWS INTO EMP**

**INSERT INTO EMP VALUES**

**('000010','CHRISTINE','I','HAAS','A00','3978', to\_date('1965-01-01','yyyy-mm-dd'), 'PRES ',18,'F',to\_date('1933-08-24','yyyy-mm-dd'), +0052750.00,+0001000.00,+0004220.00);**

**INSERT INTO EMP VALUES ('000020','MICHAEL','L','THOMPSON','B01','3476',to\_date('1973-10-10','yyyy-mm-dd'), 'MANAGER ',18,'M',to\_date('1948-02-02','yyyy-mm-dd'),+0041250.00,+0000800.00,+0003300.00);INSERT INTO EMP VALUES ('000030','SALLY','A','KWAN','C01','4738',to\_date('1975-04-05','yyyy-mm-dd'), 'MANAGER ',20,'F',to\_date('1941-05-11','yyyy-mm-dd'),+0038250.00,+0000800.00,+0003060.00);INSERT INTO EMP VALUES ('000050','JOHN','B','GEYER','E01','6789',to\_date('1949-08-17','yyyy-mm-dd'), 'MANAGER ',16,'M',to\_date('1925-09-15','yyyy-mm-dd'),+0040175.00,+0000800.00,+0003214.00);INSERT INTO EMP VALUES ('000060','IRVING','F','STERN','D11','6423',to\_date('1973-09-14','yyyy-mm-dd'), 'MANAGER ',16,'M',to\_date('1945-07-07','yyyy-mm-dd'),+0032250.00,+0000500.00,+0002580.00);INSERT INTO EMP VALUES ('000070','EVA','D','PULASKI','D21','7831',to\_date('1980-09-30','yyyy-mm-dd'), 'MANAGER ',16,'F',to\_date('1953-05-26','yyyy-mm-dd'),+0036170.00,+0000700.00,+0002893.00);INSERT INTO EMP VALUES ('000090','EILEEN','W','HENDERSON','E11','5498',to\_date('1970-08-15','yyyy-mm-dd'), 'MANAGER ',16,'F',to\_date('1941-05-15','yyyy-mm-dd'),+0029750.00,+0000600.00,+0002380.00);INSERT INTO EMP VALUES ('000100','THEODORE','Q','SPENSER','E21','0972',to\_date('1980-06-19','yyyy-mm-dd'), 'MANAGER ',14,'M',to\_date('1956-12-18','yyyy-mm-dd'),+0026150.00,+0000500.00,+0002092.00);INSERT INTO EMP VALUES ('000110','VINCENZO','G','LUCCHESSI','A00','3490',to\_date('1958-05-16','yyyy-mm-dd'), 'SALESREP',19,'M',to\_date('1929-11-05','yyyy-mm-dd'),+0046500.00,+0000900.00,+0003720.00);INSERT INTO EMP VALUES ('000120','SEAN',' ','O''CONNELL','A00','2167',to\_date('1963-12-05','yyyy-mm-dd'), 'CLERK ',14,'M',to\_date('1942-10-18','yyyy-mm-dd'),+0029250.00,+0000600.00,+0002340.00);INSERT INTO EMP VALUES ('000130','DOLORES','M','QUINTANA','C01','4578',to\_date('1971-07-28','yyyy-mm-dd'), 'ANALYST ',16,'F',to\_date('1925-09-15','yyyy-mm-dd'),+0023800.00,+0000500.00,+0001904.00);INSERT INTO EMP VALUES ('000140','HEATHER','A','NICHOLLS','C01','1793',to\_date('1976-12-15','yyyy-mm-dd'), 'ANALYST ',18,'F',to\_date('1946-01-19','yyyy-mm-dd'),+0028420.00,+0000600.00,+0002274.00);INSERT INTO EMP VALUES ('000150','BRUCE',' ','ADAMSON','D11','4510',to\_date('1972-02-12','yyyy-mm-dd'), 'DESIGNER',16,'M',to\_date('1947-05-17','yyyy-mm-dd'),+0025280.00,+0000500.00,+0002022.00);INSERT INTO EMP VALUES ('000160','ELIZABETH','R','PIANKA','D11','3782',to\_date('1977-10-11','yyyy-mm-dd'), 'DESIGNER',17,'F',to\_date('1955-04-12','yyyy-mm-dd'),+0022250.00,+0000400.00,+0001780.00);INSERT INTO EMP VALUES ('000170','MASATOSHI','J','YOSHIMURA','D11','2890',to\_date('1978-09-15','yyyy-mm-dd'), 'DESIGNER',16,'M',to\_date('1951-01-05','yyyy-mm-dd'),+0024680.00,+0000500.00,+0001974.00);INSERT INTO EMP VALUES ('000180','MARILYN','S','SCOUTTEN','D11','1682',to\_date('1973-07-07','yyyy-mm-dd'), 'DESIGNER',17,'F',to\_date('1949-02-21','yyyy-mm-dd'),+0021340.00,+0000500.00,+0001707.00);INSERT INTO EMP VALUES ('000190','JAMES','H','WALKER','D11','2986',to\_date('1974-07-26','yyyy-mm-dd'), 'DESIGNER',16,'M',to\_date('1952-06-25','yyyy-mm-dd'),+0020450.00,+0000400.00,+0001636.00);INSERT INTO EMP VALUES ('000200','DAVID',' ','BROWN','D11','4501',to\_date('1966-03-03','yyyy-mm-dd'), 'DESIGNER',16,'M',to\_date('1941-05-29','yyyy-mm-dd'),+0027740.00,+0000600.00,+0002217.00);INSERT INTO EMP VALUES ('000210','WILLIAM','T','JONES','D11','0942',to\_date('1979-04-11','yyyy-mm-dd'), 'DESIGNER',17,'M',to\_date('1953-02-23','yyyy-mm-dd'),+0018270.00,+0000400.00,+0001462.00);INSERT INTO EMP VALUES ('000220','JENNIFER','K','LUTZ','D11','0672',to\_date('1968-08-29','yyyy-mm-dd'), 'DESIGNER',18,'F',to\_date('1948-03-19','yyyy-mm-dd'),+0029840.00,+0000600.00,+0002387.00);INSERT INTO EMP VALUES ('000230','JAMES','J','JEFFERSON','D21','2094',to\_date('1966-11-21','yyyy-mm-dd'), 'CLERK ',14,'M',to\_date('1935-05-30','yyyy-mm-dd'),+0022180.00,+0000400.00,+0001774.00);INSERT INTO EMP VALUES ('000240','SALVATORE','M','MARINO','D21','3780',to\_date('1979-12-05','yyyy-mm-dd'), 'CLERK ',17,'M',to\_date('1954-03-31','yyyy-mm-dd'),+0028760.00,+0000600.00,+0002301.00);INSERT INTO EMP VALUES ('000250','DANIEL','S','SMITH','D21','0961',to\_date('1969-10-30','yyyy-mm-dd'), 'CLERK ',15,'M',to\_date('1939-11-12','yyyy-mm-dd'),+0019180.00,+0000400.00,+0001534.00);INSERT INTO EMP VALUES ('000260','SYBIL','P','JOHNSON','D21','8953',to\_date('1975-09-11','yyyy-mm-dd'), 'CLERK ',16,'F',to\_date('1936-10-05','yyyy-mm-dd'),+0017250.00,+0000300.00,+0001380.00);INSERT INTO EMP VALUES ('000270','MARIA','L','PEREZ','D21','9001',to\_date('1980-09-30','yyyy-mm-dd'), 'CLERK ',15,'F',to\_date('1953-05-26','yyyy-mm-dd'),+0027380.00,+0000500.00,+0002190.00);INSERT INTO EMP VALUES ('000280','ETHEL','R','SCHNEIDER','E11','8997',to\_date('1967-03-24','yyyy-mm-dd'), 'OPERATOR',17,'F',to\_date('1936-03-28','yyyy-mm-dd'),+0026250.00,+0000500.00,+0002100.00);INSERT INTO EMP VALUES ('000290','JOHN','R','PARKER','E11','4502',to\_date('1980-05-30','yyyy-mm-dd'), 'OPERATOR',12,'M',to\_date('1946-07-09','yyyy-mm-dd'),+0015340.00,+0000300.00,+0001227.00);INSERT INTO EMP VALUES ('000300','PHILIP','X','SMITH','E11','2095',to\_date('1972-06-19','yyyy-mm-dd'), 'OPERATOR',14,'M',to\_date('1936-10-27','yyyy-mm-dd'),+0017750.00,+0000400.00,+0001420.00);INSERT INTO EMP VALUES ('000310','MAUDE','F','SETRIGHT','E11','3332',to\_date('1964-09-12','yyyy-mm-dd'), 'OPERATOR',12,'F',to\_date('1931-04-21','yyyy-mm-dd'),+0015900.00,+0000300.00,+0001272.00);INSERT INTO EMP VALUES ('000320','RAMLAL','V','MEHTA','E21','9990',to\_date('1965-07-07','yyyy-mm-dd'), 'FIELDREP',16,'M',to\_date('1932-08-11','yyyy-mm-dd'),+0019950.00,+0000400.00,+0001596.00);INSERT INTO EMP VALUES ('000330','WING',' ','LEE','E21','2103',to\_date('1976-02-23','yyyy-mm-dd'), 'FIELDREP',14,'M',to\_date('1941-07-18','yyyy-mm-dd'),+0025370.00,+0000500.00,+0002030.00);INSERT INTO EMP VALUES ('000340','JASON','R','GOUNOT','E21','5698',to\_date('1947-05-05','yyyy-mm-dd'), 'FIELDREP',16,'M',to\_date('1926-05-17','yyyy-mm-dd'),+0023840.00,+0000500.00,+0001907.00);-- END OF INSERT FOR TABLE EMP**

**-- /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/-- /\* INSERT DATA INTO TABLE DEPT \*/-- /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/-- DELETE ALL ROWS FROM DEPT --DELETE FROM DEPT;-- INSERT NEW ROWS INTO DEPT --INSERT INTO DEPT VALUES ('A00','SPIFFY COMPUTER SERVICE DIV.','000010','A00',NULL);INSERT INTO DEPT VALUES ('B01','PLANNING','000020','A00',NULL);INSERT INTO DEPT VALUES ('C01','INFORMATION CENTER','000030','A00',NULL);INSERT INTO DEPT VALUES ('D01','DEVELOPMENT CENTER',NULL,'A00',NULL);INSERT INTO DEPT VALUES ('D11','MANUFACTURING SYSTEMS','000060','D01',NULL);INSERT INTO DEPT VALUES ('D21','ADMINISTRATION SYSTEMS','000070','D01',NULL);INSERT INTO DEPT VALUES ('E01','SUPPORT SERVICES','000050','A00',NULL);INSERT INTO DEPT VALUES ('E11','OPERATIONS','000090','E01',NULL);INSERT INTO DEPT VALUES ('E21','SOFTWARE SUPPORT','000100','E01',NULL);**

**-- END OF INSERT FOR TABLE DEPT**

**DELETE FROM PROJ;INSERT INTO PROJ VALUES ('AD3100','ADMIN SERVICES','D01','000010',+006.50, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),' ');INSERT INTO PROJ VALUES ('AD3110','GENERAL ADMIN SYSTEMS','D21','000070',+006.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'AD3100');INSERT INTO PROJ VALUES ('AD3111','PAYROLL PROGRAMMING','D21','000230',+002.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'AD3110');INSERT INTO PROJ VALUES ('AD3112','PERSONNEL PROGRAMMING','D21','000250',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'AD3110');INSERT INTO PROJ VALUES ('AD3113','ACCOUNT PROGRAMMING','D21','000270',+002.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'AD3110');INSERT INTO PROJ VALUES ('IF1000','QUERY SERVICES','C01','000030',+002.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),NULL);INSERT INTO PROJ VALUES ('IF2000','USER EDUCATION','C01','000030',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),NULL);INSERT INTO PROJ VALUES ('MA2100','WELD LINE AUTOMATION','D01','000010',+012.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),NULL);INSERT INTO PROJ VALUES ('MA2110','W L PROGRAMMING','D11','000060',+009.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'MA2100');INSERT INTO PROJ VALUES ('MA2111','W L PROGRAM DESIGN','D11','000220',+002.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-12-01','yyyy-mm-dd'),'MA2110');INSERT INTO PROJ VALUES ('MA2112','W L ROBOT DESIGN','D11','000150',+003.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-12-01','yyyy-mm-dd'),'MA2110');INSERT INTO PROJ VALUES ('MA2113','W L PROD CONT PROGS','D11','000160',+003.00, to\_date('1982-02-15','yyyy-mm-dd'),to\_date('1982-12-01','yyyy-mm-dd'),'MA2110');INSERT INTO PROJ VALUES ('OP1000','OPERATION SUPPORT','E01','000050',+006.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),NULL);INSERT INTO PROJ VALUES ('OP1010','OPERATION','E11','000090',+005.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'OP1000');INSERT INTO PROJ VALUES ('OP2000','GEN SYSTEMS SERVICES','E01','000050',+005.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),NULL);INSERT INTO PROJ VALUES ('OP2010','SYSTEMS SUPPORT','E21','000100',+004.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'OP2000');INSERT INTO PROJ VALUES ('OP2011','SCP SYSTEMS SUPPORT','E21','000320',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'OP2010');INSERT INTO PROJ VALUES ('OP2012','APPLICATIONS SUPPORT','E21','000330',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'OP2010');INSERT INTO PROJ VALUES ('OP2013','DB/DC SUPPORT','E21','000340',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'),'OP2010');INSERT INTO PROJ VALUES ('PL2100','WELD LINE PLANNING','B01','000020',+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-09-15', 'yyyy-mm-dd'),'MA2100');-- END OF INSERT FOR TABLE PROJ**

**-- INSERT NEW ROWS INTO EMPPROJACT --**

**INSERT INTO EMPPROJACT VALUES**

**('000010','MA2100',10,+000.50, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-11-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000010','MA2110',10,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000010','AD3100',10,+000.50, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-07-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000020','PL2100',30,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-09-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000030','IF1000',10,+000.50, to\_date('1982-06-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000030','IF2000',10,+000.50, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000050','OP1000',10,+000.25, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000050','OP2010',10,+000.75, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000070','AD3110',10,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000090','OP1010',10,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000100','OP2010',10,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000110','MA2100',20,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-03-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000130','IF1000',90,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-10-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000130','IF1000',100,+000.50,to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000140','IF1000',90,+000.50, to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000140','IF2000',100,+001.00,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-03-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000140','IF2000',100,+000.50,to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-07-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000140','IF2000',110,+000.50,to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-07-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000140','IF2000',110,+000.50,to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000150','MA2112',60,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-07-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000150','MA2112',180,+001.00,to\_date('1982-07-15','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000160','MA2113',60,+001.00, to\_date('1982-07-15','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000170','MA2112',60,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-06-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000170','MA2112',70,+001.00, to\_date('1982-06-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000170','MA2113',80,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000180','MA2113',70,+001.00, to\_date('1982-04-01','yyyy-mm-dd'),to\_date('1982-06-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000190','MA2112',70,+001.00, to\_date('1982-02-01','yyyy-mm-dd'),to\_date('1982-10-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000190','MA2112',80,+001.00, to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-10-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000200','MA2111',50,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-06-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000200','MA2111',60,+001.00, to\_date('1982-06-15','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000210','MA2113',80,+000.50, to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000210','MA2113',180,+000.50,to\_date('1982-10-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000220','MA2111',40,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000230','AD3111',60,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-03-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000230','AD3111',60,+000.50, to\_date('1982-03-15','yyyy-mm-dd'),to\_date('1982-04-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000230','AD3111',70,+000.50, to\_date('1982-03-15','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000230','AD3111',80,+000.50, to\_date('1982-04-15','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000230','AD3111',180,+001.00,to\_date('1982-10-15','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000240','AD3111',70,+001.00, to\_date('1982-02-15','yyyy-mm-dd'),to\_date('1982-09-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000240','AD3111',80,+001.00, to\_date('1982-09-15','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',60,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',60,+000.50, to\_date('1982-02-01','yyyy-mm-dd'),to\_date('1982-03-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',60,+000.50, to\_date('1982-12-01','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',60,+001.00, to\_date('1983-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',70,+000.50, to\_date('1982-02-01','yyyy-mm-dd'),to\_date('1982-03-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',70,+001.00, to\_date('1982-03-15','yyyy-mm-dd'),to\_date('1982-08-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',70,+000.25, to\_date('1982-08-15','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',80,+000.25, to\_date('1982-08-15','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',80,+000.50, to\_date('1982-10-15','yyyy-mm-dd'),to\_date('1982-12-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000250','AD3112',180,+000.50,to\_date('1982-08-15','yyyy-mm-dd'),to\_date('1983-01-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',70,+000.50, to\_date('1982-06-15','yyyy-mm-dd'),to\_date('1982-07-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',70,+001.00, to\_date('1982-07-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',80,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-03-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',80,+000.50, to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-04-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',180,+000.50,to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-04-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',180,+001.00,to\_date('1982-04-15','yyyy-mm-dd'),to\_date('1982-06-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000260','AD3113',180,+000.50,to\_date('1982-06-01','yyyy-mm-dd'),to\_date('1982-07-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',60,+000.50, to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-04-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',60,+001.00, to\_date('1982-04-01','yyyy-mm-dd'),to\_date('1982-09-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',60,+000.25, to\_date('1982-09-01','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',70,+000.75, to\_date('1982-09-01','yyyy-mm-dd'),to\_date('1982-10-15','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',70,+001.00, to\_date('1982-10-15','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',80,+001.00, to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1982-03-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000270','AD3113',80,+000.50, to\_date('1982-03-01','yyyy-mm-dd'),to\_date('1982-04-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000280','OP1010',130,+001.00,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000290','OP1010',130,+001.00,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000300','OP1010',130,+001.00,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000310','OP1010',130,+001.00,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000320','OP2011',140,+000.75,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000320','OP2011',150,+000.25,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000330','OP2012',140,+000.25,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000330','OP2012',160,+000.75,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000340','OP2013',140,+000.50,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));INSERT INTO EMPPROJACT VALUES ('000340','OP2013',170,+000.50,to\_date('1982-01-01','yyyy-mm-dd'),to\_date('1983-02-01','yyyy-mm-dd'));-- END OF INSERTS FOR TABLE EMPPROJACT-- /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/-- /\* COMMIT UPDATES \*/-- /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**COMMIT;**

## 1.3 create\_view\_project.sql Script

**CREATE VIEW PROJECT**

**( PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,**

**PRSTDATE, PRENDATE, MAJPROJ )**

**AS**

**SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,**

**PRSTDATE, PRENDATE, MAJPROJ**

**FROM PROJ;**

**CREATE VIEW EMPLOYEE**

**( EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,**

**PHONENO, HIREDATE, JOB, EDLEVEL, SEX,**

**BIRTHDATE, SALARY, BONUS, COMM )**

**AS**

**SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,**

**PHONENO, HIREDATE, JOB, EDLEVEL, SEX,**

**BIRTHDATE, SALARY, BONUS, COMM**

**FROM EMP**

**WHERE EMPNO <= '000340';**

**CREATE VIEW DEPARTMENT**

**( DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION )**

**AS**

**SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION**

**FROM DEPT**

**WHERE DEPTNO IN ( 'A00', 'B01', 'C01', 'D01', 'D11', 'D21', 'E01',**

**'E11', 'E21' );**

**CREATE VIEW EMP\_ACT ( EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE )**

**AS**

**SELECT EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE**

**FROM EMPPROJACT3**

**COMMIT;**

## 1.4 Sample Data and Table Structure

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EMP (Employee) Table** | | | | | | | | | | | | | |
| **EMPNO** | **FIRSTNME** | **MIDINIT** | **LASTNAME** | **WORKDEPT** | **PHONENO** | **HIREDATE** | **JOB** | **EDLEVEL** | **SEX** | **BIRTHDATE** | **SALARY** | **BONUS** | **COMM** |
| **CHAR(6)** | **VARCHAR(12)** | **CHAR(1)** | **VARCHAR(15)** | **CHAR(3)** | **CHAR(4)** | **DATE** | **CHAR(8)** | **SMALLINT** | **CHAR(1)** | **DATE** | **DECIMAL(9,2)** | **DECIMAL(9,2)** | **DECIMAL(9,2)** |
| '000010' | 'CHRISTINE' | 'I'' | 'HAAS' | 'A00' | '3978' | '1965-01-01' | 'PRES ' | 18 | 'F' | '1933-08-24' | +0052750.00 | +0001000.00 | +0004220.00 |
| '000020' | 'MICHAEL' | 'L' | 'THOMPSON' | 'B01' | '3476 | '1973-10-10' | 'MANAGER | 18 | 'M' | '1948-02-02' | +0041250.00 | +0000800.00 | +0003300.00 |
| '000030' | 'SALLY' | 'A' | 'KWAN | C01' | 4738' | '1975-04-05' | 'MANAGER ', | 20 | 'F' | '1941-05-11' | +0038250.00 | +0000800.00 | +0003060.00 |
| '000050' | 'JOHN' | 'B' | 'GEYER' | 'E01' | '6789' | '1949-08-17' | 'MANAGER ' | 16 | 'M' | '1925-09-15' | +0040175.00 | +0000800.00 | +0003214.00 |
| '000060' | 'IRVING' | 'F' | 'STERN' | 'D11' | '6423' | '1973-09-14' | 'MANAGER ' | 16 | 'M' | '1945-07-07' | +0032250.00 | +0000500.00 | +0002580.00 |
| '000070' | 'EVA' | 'D' | 'PULASKI | 'D21' | '7831' | '1980-09-30 | 'MANAGER ' | 16 | 'F' | '1953-05-26' | ,+0036170.00 | ,+0000700.00 | ,+0002893.00 |
| '000090' | 'EILEEN' | 'W' | 'HENDERSON | 'E11' | '5498 | '1970-08-15' | 'MANAGER' | 16 | 'F' | '1941-05-15' | +0029750.00 | +0000600.00 | +0002380.00 |
| '000100' | 'THEODORE' | 'Q' | 'SPENSER', | 'E21' | '0972' | '1980-06-19' | 'MANAGER ' | 14 | 'M' | '1956-12-18' | +0026150.00 | +0000500.00 | +0002092.00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DEP (Department) Table** | | | | |
| **DEPTNO** | **DEPTNAME** | **MGRNO** | **ADMRDEPT** | **LOCATION** |
| **CHAR(3)** | **VARCHAR(29)** | **CHAR(6)** | **CHAR(3)** | **CHAR(16)** |
| 'A00' | 'SPIFFY COMPUTER SERVICE DIV. | '000010' | 'A00' | NULL |
| 'B01' | 'PLANNING' | '000020' | 'A00' | NULL |
| 'C01' | 'INFORMATION CENTER' | '000030' | 'A00' | NULL |
| 'D01' | 'DEVELOPMENT CENTER' | NULL | 'A00' | NULL |
| 'D11' | 'MANUFACTURING SYSTEMS' | '000060' | 'D01' | NULL |
| 'D21' | 'ADMINISTRATION SYSTEMS' | '000070' | ,'D01' | NULL |
| 'E01' | 'SUPPORT SERVICES' | '000050' | 'A00' | NULL |
| 'E11' | 'OPERATIONS' | '000090' | E01' | NULL |
| 'E21' | 'SOFTWARE SUPPORT' | '000100' | 'E01' | NULL |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PROJ (Project) Table** | | | | | | | |
| **PROJNO** | **PROJNAME** | **DEPTNO** | **RESPEMP** | **PRSTAFF** | **PRSTDATE** | **PRENDATE** | **MAJPROJ** |
| **CHAR(6)** | **VARCHAR(24)** | **CHAR(3)** | **CHAR(6)** | **DECIMAL(5,2)** | **DATE** | **DATE** | **CHAR(6)** |
| 'AD3100' | 'ADMIN SERVICES' | 'D01' | '000010' | +006.50 | '1982-01-01' | '1983-02-01' | ' ' |
| 'AD3110' | 'GENERAL ADMIN SYSTEMS' | 'D21' | '000070' | +006.00 | '1982-01-01' | '1983-02-01' | 'AD3100' |
| 'AD3111' | 'PAYROLL PROGRAMMING' | 'D21' | '000230' | +002.00 | '1982-01-01' | '1983-02-01' | 'AD3110' |
| 'AD3112' | 'PERSONNEL PROGRAMMING' | 'D21' | '000250' | +001.00 | '1982-01-01' | '1983-02-01' | AD3110' |
| 'AD3113' | 'ACCOUNT PROGRAMMING' | 'D21' | '000270' | +002.00, | '1982-01-01' | '1983-02-01' | 'AD3110' |
| 'IF1000' | 'QUERY SERVICES' | 'C01' | '000030' | +002.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'IF2000' | 'USER EDUCATION' | 'C01' | '000030' | +001.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'MA2100' | 'WELD LINE AUTOMATION' | 'D01' | '000010' | +012.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'MA2110' | 'W L PROGRAMMING' | 'D11' | '000060' | +009.00 | '1982-01-01' | '1983-02-01' | 'MA2100' |
| 'MA2111' | 'W L PROGRAM DESIGN' | 'D11' | '000220' | +002.00 | '1982-01-01' | '1982-12-01' | 'MA2110 |
| 'MA2112 | 'W L ROBOT DESIGN' | 'D11 | '000150' | +003.00 | '1982-01-01' | '1982-12-01' | 'MA2110' |
| 'MA2113' | 'W L PROD CONT PROGS' | 'D11' | '000160' | +003.00, | '1982-02-15' | '1982-12-01' | 'MA2110' |
| 'OP1000' | 'OPERATION SUPPORT' | 'E01' | '000050' | +006.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'OP1010' | 'OPERATION' | 'E11' | '000090' | +005.00 | '1982-01-01' | '1983-02-01' | 'OP1000' |
| 'OP2000' | 'GEN SYSTEMS SERVICES' | 'E01 | '000050' | +005.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'OP2010' | 'SYSTEMS SUPPORT' | 'E21' | '000100' | +004.00 | '1982-01-01' | '1983-02-01' | 'OP2000' |
| 'OP2011' | 'SCP SYSTEMS SUPPORT' | 'E21' | '000320' | +001.00 | '1982-01-01' | '1983-02-01' | 'OP2010' |
| 'OP2012' | 'APPLICATIONS SUPPORT' | 'E21' | '000330' | +001.00 | '1982-01-01' | '1983-02-01' | 'OP2010' |
| 'OP2013' | 'DB/DC SUPPORT' | 'E21' | '000340' | +001.00 | '1982-01-01' | '1983-02-01' | 'OP2010' |
| 'PL2100' | 'WELD LINE PLANNING' | 'B01' | '000020' | +001.00 | '1982-01-01' | '1982-09-15' | 'MA2100' |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPPROJACT Table** | | | | | |
| **EMPNO** | **PROJNO** | **ACTNO** | **EMPTIME** | **EMSTDATE** | **EMENDATE** |
| **CHAR(6)** | **CHAR(6)** | **SMALLINT** | **DECIMAL(5,2)** | **DATE** | **DATE** |
| '000010' | 'MA2100' | 10 | +000.50 | '1982-01-01' | '1982-11-01' |
| '000010' | 'MA2110' | 10 | +001.00 | '1982-01-01' | '1983-02-01' |
| '000010' | 'AD3100' | 10 | +000.50 | '1982-01-01' | '1982-07-01' |
| '000020' | 'PL2100' | 30 | +001.00 | '1982-01-01' | '1982-09-15' |
| '000030' | 'IF1000' | 10 | +000.50 | '1982-06-01' | '1983-01-01' |
| '000030' | 'IF2000' | 10 | +000.50 | '1982-01-01' | '1983-01-01' |
| '000050' | 'OP1000' | 10 | +000.25 | '1982-01-01' | '1983-02-01' |
| '000050' | 'OP2010' | 10 | +000.75 | '1982-01-01' | '1983-02-01' |
| '000070' | 'AD3110' | 10 | +001.00 | '1982-01-01' | '1983-02-01' |
| '000090' | 'OP1010' | 10 | +001.00 | '1982-01-01' | 1983-02-01' |
| '000100' | 'OP2010' | 10 | +001.00 | '1982-01-01' | '1983-02-01' |
| '000110' | 'MA2100' | 20 | +001.00 | 1982-01-01' | '1982-03-01' |
| '000130' | 'IF1000' | 90 | +001.00 | '1982-01-01' | '1982-10-01' |
| '000130' | 'IF1000' | 100 | +000.50 | '1982-10-01' | '1983-01-01' |
| '000140' | 'IF1000' | 90 | +000.50 | '1982-10-01' | '1983-01-01' |
| '000140' | 'TF2000' | 100 | +001.00 | '1982-01-01' | '1982-03-01' |
| '000140' | 'IF2000' | 100 | +000.50 | '1982-03-01' | '1982-07-01' |
| '000140' | 'IF2000' | 110 | +000.50 | '1982-03-01' | '1982-07-01' |
| '000140' | 'IF2000' | 110 | +000.50 | '1982-10-01' | '1983-01-01' |

## 1.5 Relationship between DEPT and EMP

**Before designing the structure and relationship between tables one should understand the concepts of normalization discussed in Assignment 2 Theory and the business-design rules (requirements) affecting the data stored in the table. For example, a work department (DEPT row) may have many employees (EMP rows), and an employee (EMP row) have only one work department. If you review the sample department names you see that the role of the department table is to assigned an employee to a project function called the work department. Though the sample data does not show it (look at insert-project.sql), there are three employees who have been assigned the work department with the DEPTNO of 'A00'**

**The first rule of database normalization states that one should store repeating information into two separated tables, or to design a one--to-many relationship. This is the reason why the DEPT and EMP tables were separated and designed as two separate tables. The DEPT table is called the ONE (or parent) table and the EMP table is called the MANY table (or many) table. A relationship is created when the EMP table contains the WORKDEP column of the DEPT table that matches the DEPTNO column. See this relationship line in the next graphic.**

## 1.6 Relationship between DEPT and PROJ

**A work department role (DEPTNO) may be assigned to multiple, different project tasks (PROJ row identified by PROJNO, but a project may be only assigned one work department role. For those who have some experience with project management, a project has many tasks. In these examples row displayed below, it would seem that Project MA2 has several different subtasks as identified by MA2100, MA2110, MA2111, MA2112, and MA2113. Project task MA2100 is assigned to department 'D01' which is scheduled to start on '1982-01-01' and end on '1983-02-01'. Its prerequisite MAJPROJ is NULL which means that this subtask is the start of the project. Project task MA2110, MA2111, MA2112, and MA2113 is assigned to department 'D11' which is scheduled to start after the MAJPROJ is 'MA2100' starts or is completed. A project can only be assigned to one department, which is managed by one employee labeled RESPEMP**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PROJ (Project) Table** | | | | | | | | |
| **PROJNO** | **PROJNAME** | | **DEPTNO** | **RESPEMP** | **PRSTAFF** | **PRSTDATE** | **PRENDATE** | **MAJPROJ** |
| 'MA2100' | | 'WELD LINE AUTOMATION' | 'D01' | '000010' | +012.00 | '1982-01-01' | '1983-02-01' | NULL |
| 'MA2110' | | 'W L PROGRAMMING' | 'D11' | '000060' | +009.00 | '1982-01-01' | '1983-02-01' | 'MA2100' |
| 'MA2111' | | 'W L PROGRAM DESIGN' | 'D11' | '000220' | +002.00 | '1982-01-01' | '1982-12-01' | 'MA2110 |
| 'MA2112 | | 'W L ROBOT DESIGN' | 'D11 | '000150' | +003.00 | '1982-01-01' | '1982-12-01' | 'MA2110' |
| 'MA2113' | | 'W L PROD CONT PROGS' | 'D11' | '000160' | +003.00, | '1982-02-15' | '1982-12-01' | 'MA2110' |

## 1.7 Relationships between EMP and PROJ with EMPPROJACT

**Table EMPPROJACT represents the typical structure of a business transaction table. Each EMPPROJACT row stores data for one transaction to assign employee time to a project. The EMPNO column stores data concerning who was responsible for the transaction, the PROJNO provide data for the project and indirectly the DEPT or activity actually performed. The employee project activity table (EMPPROJACT) accumulates project time charges. Each row presents charges for one employee, for one project and to be charged for only one ACTNO (account number). Therefore, an employee may have many charges and a project/sub-project may have many charges.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPPROJACT Table** | | | | | |
| **EMPNO** | **PROJNO** | **ACTNO** | **EMPTIME** | **EMSTDATE** | **EMENDATE** |
| **CHAR(6)** | **CHAR(6)** | **SMALLINT** | **DECIMAL(5,2)** | **DATE** | **DATE** |
| '000010' | 'MA2100' | 10 | +000.50 | '1982-01-01' | '1982-11-01' |
| '000010' | 'MA2110' | 10 | +001.00 | '1982-01-01' | '1983-02-01' |

**The relationships between EMP and PROJ with EMPPROJACT is an example of a many-to-many relationship. An employee can have many time transactions assigned to a project, and a project can have many employees working on a project. EMPPROJACT is an example of third table which is often called an intersection table or help table. Many-to-many relationships exist in the real world, but impossible to represent using two tables. See the normalization handout for more details.**

**In the following diagram notice the relationship between the four tables. A table relationship occurs when two tables have a common field, e.g., EMPNO, DEPTNO, and PROJNO.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EMP (Employee) Table** | | | | | | | | | | | | | |
| **EMPNO** | **FIRSTNME** | **MIDINIT** | **LASTNAME** | **WORKDEPT** | **PHONENO** | **HIREDATE** | **JOB** | **EDLEVEL** | **SEX** | **BIRTHDATE** | **SALARY** | **BONUS** | **COMM** |
| **CHAR(6)** | **VARCHAR(12)** | **CHAR(1)** | **VARCHAR(15)** | **CHAR(3)** | **CHAR(4)** | **DATE** | **CHAR(8)** | **SMALLINT** | **CHAR(1)** | **DATE** | **DECIMAL(9,2)** | **DECIMAL(9,2)** | **DECIMAL(9,2)** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DEPT (Department) Table** | | | | |
| **DEPTNO** | **DEPTNAME** | **MGRNO** | **ADMRDEPT** | **LOCATION** |
| **CHAR(3)** | **VARCHAR(29)** | **CHAR(6)** | **CHAR(3)** | **CHAR(16)** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PROJ (Project) Table** | | | | | | | |
| **PROJNO** | **PROJNAME** | **DEPTNO** | **RESPEMP** | **PRSTAFF** | **PRSTDATE** | **PRENDATE** | **MAJPROJ** |
| **CHAR(6)** | **VARCHAR(24)** | **CHAR(3)** | **CHAR(6)** | **DECIMAL(5,2)** | **DATE** | **DATE** | **CHAR(6)** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPPROJACT Table** | | | | | |
| **EMPNO** | **PROJNO** | **ACTNO** | **EMPTIME** | **EMSTDATE** | **EMENDATE** |
| **CHAR(6)** | **CHAR(6)** | **SMALLINT** | **DECIMAL(5,2)** | **DATE** | **DATE** |

**While the previous four tables have been normalized, the tables and relationship design has poor data integrity.** **Data integrity refers to maintaining and assuring the accuracy and consistency of data over its entire life-cycle and is a critical aspect to the design. Entity integrity concerns the concept of a primary key, which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null. There are no primary keys declared. It is therefore possible that two employees may be assigned the same and duplicate EMPNO. Besides trying to determine which EMPNO '000010' gets paid, the design may also have problems for accounting to time to two different projects that have the same and duplicate PROJNO.**

**Referential integrity concerns the concept of a foreign key. The referential integrity rule states that any foreign-key value can only be in one of two states. The usual state of affairs is that the foreign-key value refers to a primary key value of some table in the database. For example, would it be logical to charge employee time in the EMPPROJACT table to an EMPNO that does not exist in the EMP table or a PROJNO in the PROJ table that also may not exist. This of course violates common sense. Common senses in DBMS is also called business rules.**

**Finally, the may be other business rules, such as range of employee salary, type of JOB, or valid data range which may be implemented by a CHECK CONSTRATINT. But, this design example is simple and will be improved in future assignments.**

# 2.0 Questions - Database Project Design

**Refer to the previous discussions and following links. Answer the following questions.**

1. Unlike storing data in an Excel Workbook, before data is stored in a relational database table what steps must occur? **Answer =>**

First design the structure of the data as tables and relationships between the tables.

Define the column name or attributes, then data type of each column must be mentioned.

.Spring 2018

2. What is a function of a data type when storing data in either a relational table column or a programming language variable? **Answer =>**

Data type used in database or programming language variables permits only particular type of data in different columns. Like a numeric data permits only numeric value, nor any alphabetic or special character.

.Spring 2018

3. Describe the concept of a ***one-to-many relationship*** using the relationship between EMP and DEPT tables. **Answer =>**

.an EMP must have to work in one DEPT and a DEPT can have many employees.Spring 2018

4. Describe the concept of a ***many-to-many relationship*** using the relationship between EMP, PROJ and EMPPROJACT tables. **Answer =>**

An employee can have several times transaction assigned to a project, a project can have many employee working on a project.

.Spring 2018

Should each and every table have a primary key? <http://stackoverflow.com/questions/840162/should-each-and-every-table-have-a-primary-key>

Advantages of having a primary key -

http://www.startingabusiness.ca/online-courses/microsoft-access/database-design-1/primary-key-advantages.htm

Choosing a Primary Key: Natural or Surrogate? - <http://www.agiledata.org/essays/keys.html>

Entity and Referential Integrity - <https://oozden.wordpress.com/2014/05/18/entity-integrity-and-referential-integrity-what-are-they/>

Benefits of Referential Integrity - http://www.info.teradata.com/HTMLPubs/DB\_TTU\_14\_00/index.html#page/Database\_Management/B035\_1094\_111A/apph.065.08.html

## 2.1 Questions - Primary Key Constraint

5. The previous EMP table has no Primary Key constraint. What are the potential problems of designing a table, similar to the previous tables, with ***no primary keys***? **Answer =>**

In relational database, the most important thing is maintain relation between different tables. Without any primary key it is not possible to design relationships between different tables.

.Spring 2018

## 2.2 Questions - Entity and Referential Integrity

6. Explain the concept of ***Entity Integrity***. **Answer =>**

Entity integrity helps the system to maintain primary key in the database. It ensures two properties for primary keys it should not match with any another primary key and it should not be null.

.Spring 2018

7. Explain the concept of ***Referential Integrity***. **Answer =>**

Referential integrity helps the system to maintain foreign key. It will ensure only one property that a foreign key must always refer to a valid primary key.

.Spring 2018

8. How is Referential Integrity enforced in SQL DDL? **Answer =>**

.Spring 2018

9. If a business requirement states that no EMPNO can be stored in the EMPPROJACT table that the EMPNO did not exist in the EMP table, how would one enforce this business rule using SQL DDL? **Answer =>**

.Spring 2018

## 2.11 Questions - Check Constraints

10. If a business requirement states that no EMP row can have salary above $250,000, how would one enforce this business rule using SQL DDL? **Answer =>**

# Requirement 3. Simple SQL Select Queries

In this requirement you are required to individually execute at the **SQL> prompt** the sample SQL statements from this presentation and document the results. The following requirements will list the topic of individual slides. All SQL statements and output need to be documented using the Windows snippit tool or other comparable tool.

Video - Introduction to SQL - SQL LESSON 1 - <https://www.youtube.com/watch?v=UeJKioNqe5w&feature=relmfu>

Video - Express Tour of SQL - PART 2 - SQL LESSON 3 - <https://www.youtube.com/watch?v=5bcLy2PT8Fk&feature=relmfu>

Video - SQL Conditions, Expressions, and Operators - PART 1 - SQL LESSON 6 - <https://www.youtube.com/watch?v=lPzW1H1nbaE&feature=relmfu>

Video - SQL Conditions, Expressions, and Operators - PART 2 - SQL LESSON 7 - <https://www.youtube.com/watch?v=3mnc7LROq-0&feature=relmfu>

Video - SQL Conditions, Expressions, and Operators - PART 3 - SQL LESSON 8 - <https://www.youtube.com/watch?v=zUAE2Ao1m-M&feature=relmfu>

Video - SQL Conditions, Expressions, and Operators - PART 4 - SQL LESSON 9 -

https://www.youtube.com/watch?v=RzS7eDHfGZk&feature=relmfu

**SELECT Statement**

The SELECT statement is used to retrieve records from the database. The basic format of the select statement is:

**SELECT** Display the contents of these column names, calculations and column functions

**FROM** Table name

**WHERE** row conditional selection

**ORDER BY** sort data using this column

**GROUP BY** subtotal by this column

[IN external database name]

**Example 1:** **SELECT \* FROM STUDENT;**

Retrieve all records and all fields will be retrieved from the STUDENT table. Each SQL statement must end with a trailing semicolon. The asterisk '\*' indicates that all columns are to be retrieved. If the table STUDENT is in the current opened database, no external database name is required.

Special note: The \* is a shorthand for selecting all columns from the result set; the alternative way of writing this is to spell out all the columns you expect in the STUDENT table, e.g. SELECT SNO, SNAME, MAJOR FROM STUDENT.

The reason people discourage SELECT \* are twofold: first of all, if you use it in production code, and the database changes for whichever reasons, your column ordering will be off, and you will be fetching results into the wrong variables. If you specify columns explicitly, you can rely on the correct number and ordering of result columns, and the query will fail loudly if the columns you specify do not match what was found in the database. It also could encourage the client programmer to expect the fields to be returned in a particular order, and with a particular cardinality. If you then extract the 'first' field from the result set and display it, the result will look correct, until someone adds or changes the column definition of the underlying table, whereupon the previously working code suddenly breaks, with neither side feeling responsible.

Another reason is performance. SELECT \* means that the DBMS has to perform an extra lookup to get the list of columns. It also means that you haven't thought about what you really need, so chances are you'll be fetching more data than you need, which, considering the typical performance bottlenecks in web applications, is something you really want to avoid. It has to retrieve more data than a focused query for the data that you actually need, so it can waste resources.

Therefore, best practice is to explicitly select those columns that are actually wanted, and retrieve them by name rather than by position.

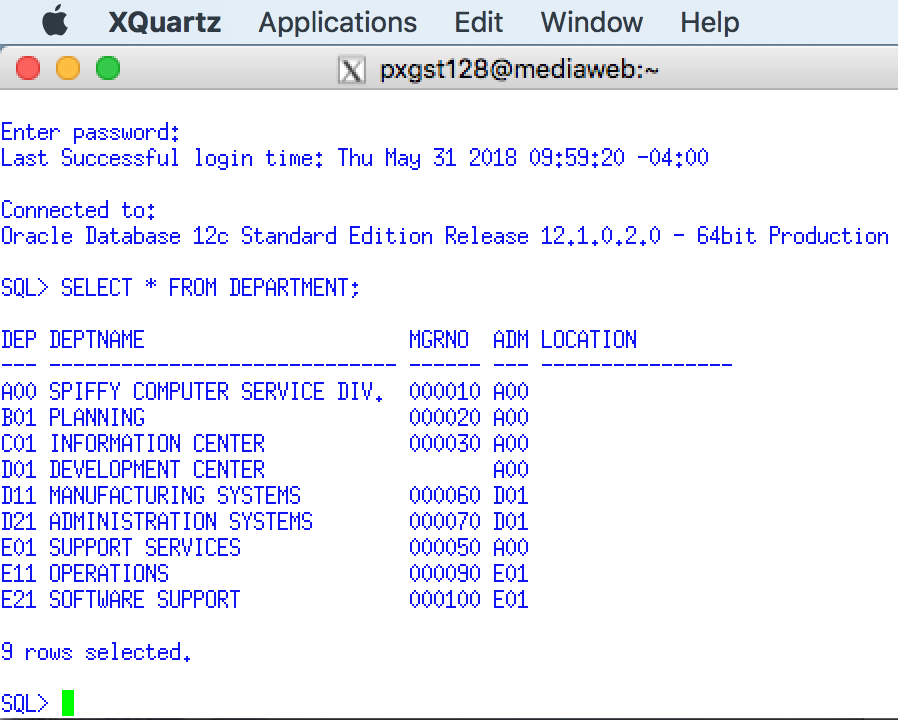
## 3.1 Basic Select Statement

**Your snipping tool documentation must clearly display your Putty Banner.**

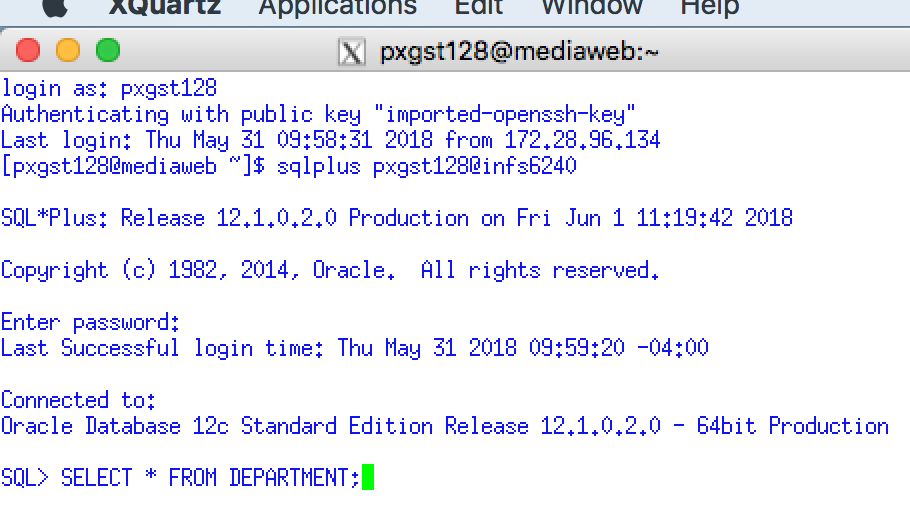
1. Retrieving All Columns, All Rows - Requirement - You need a listing of all department data. At the SQL Prompt type clear screen to erase your Putty screen. Code the following SQL Statement followed by a semicolon to execute.



Use a Snippit to document your SQL output =>



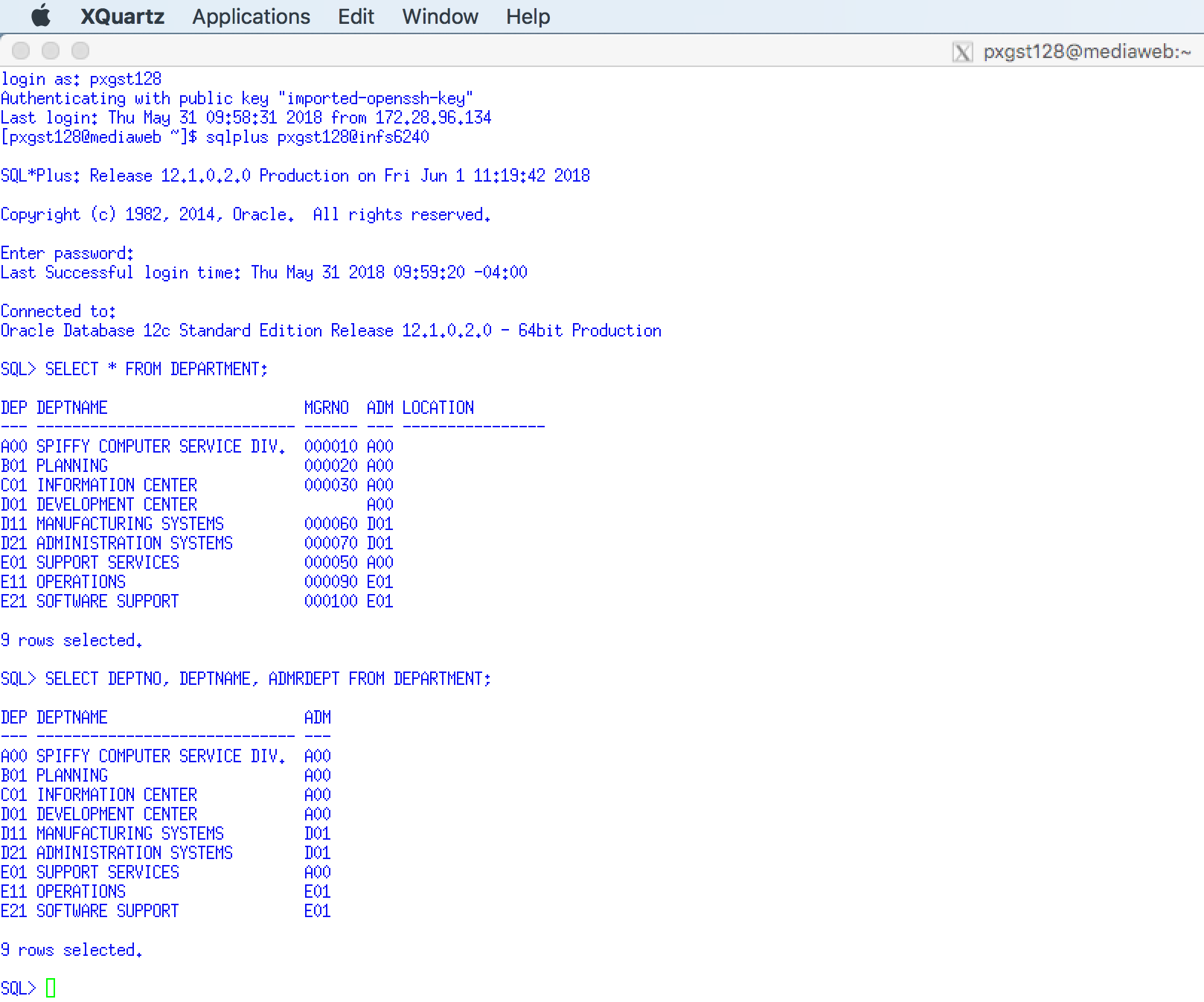
Use a Snippit to document your SQL statement =>



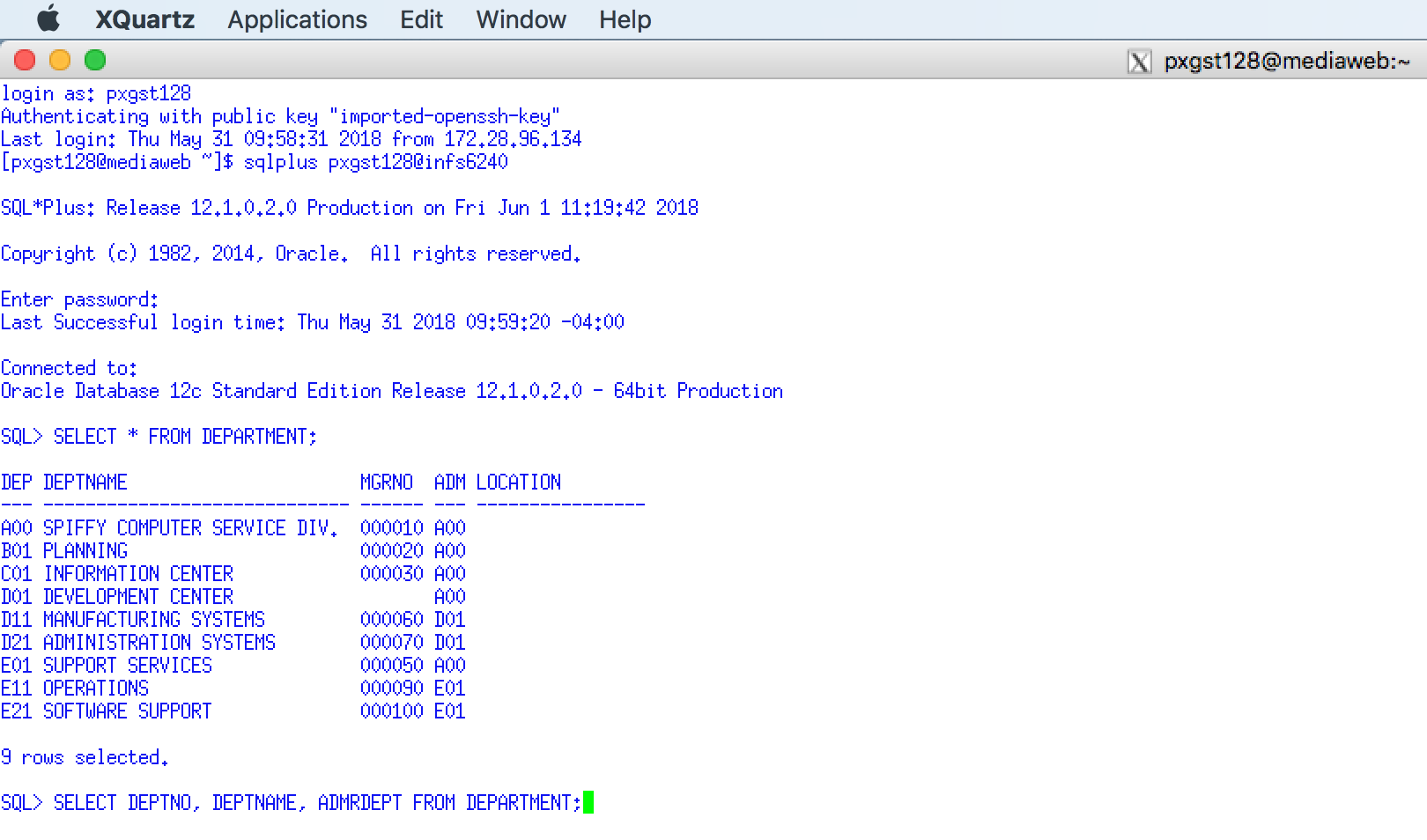
2. Selecting Specific Columns - Requirement- **For each department, display its number, its name and the department to which it reports.** Code the following SQL Statement followed by a semicolon to execute.



Use a Snippit to document your SQL output =>



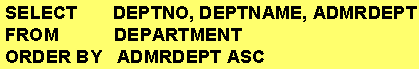
Use a Snippit to document your SQL statement =>



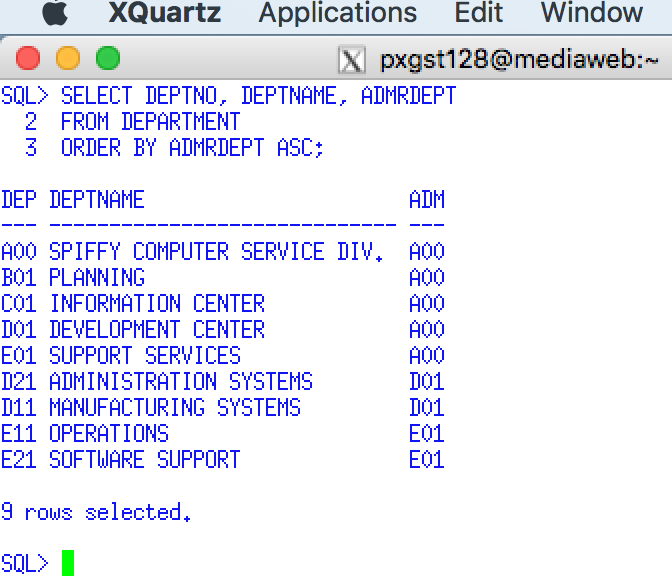
## 3.2 Order By Clause

**Your snipping tool documentation must clearly display your Putty Banner.**

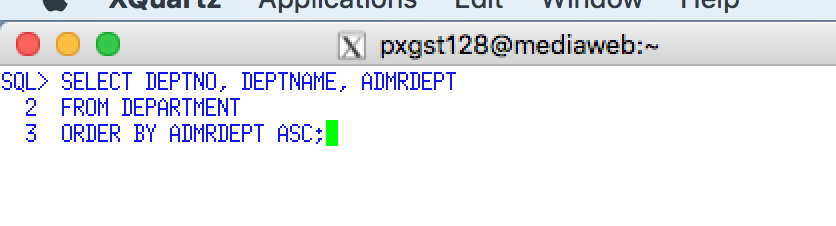
3. Select with Ordered Output - Requirement - **The listing should be sorted by the department reported to.** Code the following SQL Statement followed by a semicolon to execute.



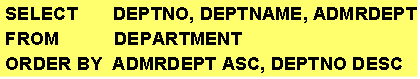
Use a Snippit to document your SQL output =>



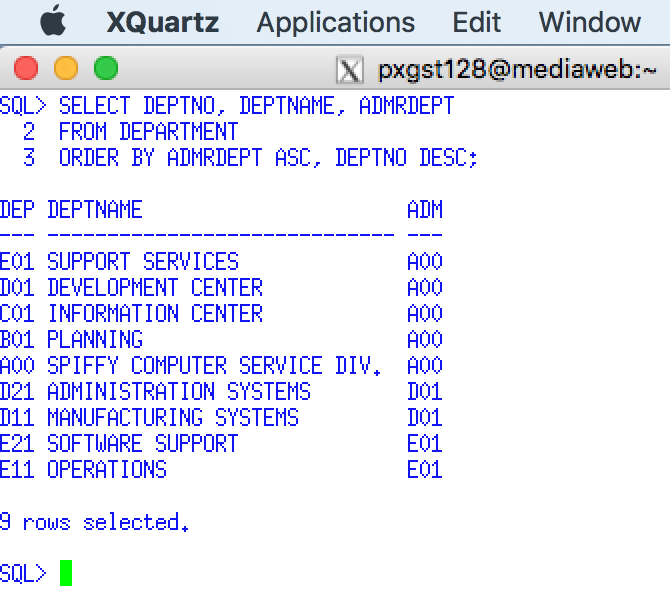
Use a Snippit to document your SQL statement =>



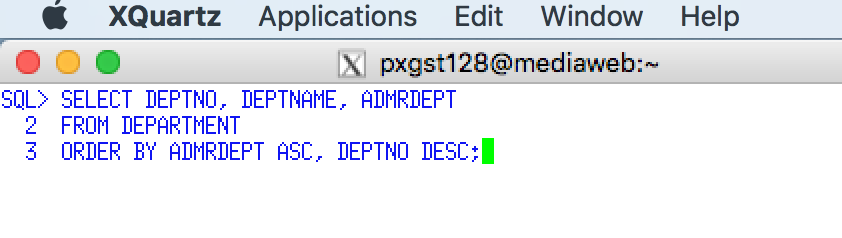
4. Select with Ordered Output - Requirement -**List the department information first according to the number to which the department reports and within that sequence sorted by descending department numbers.** Code the following SQL Statement followed by a semicolon to execute.



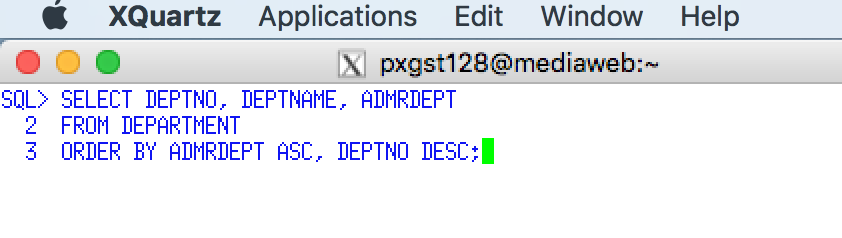
Execute this example and display the result. Use a Snippit to document your SQL output =>



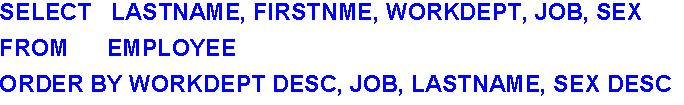
Use a Snippit to document your SQL statement =>



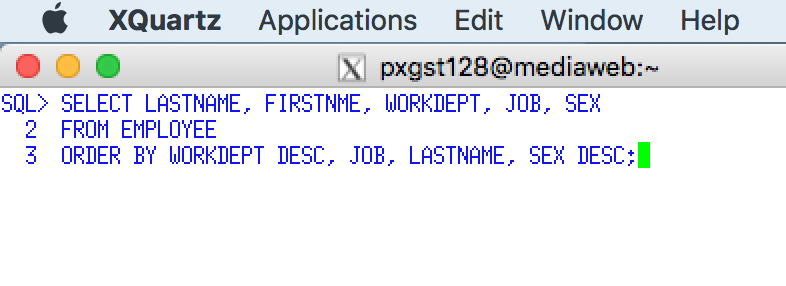
Use a Snippit to document your SQL statement =>



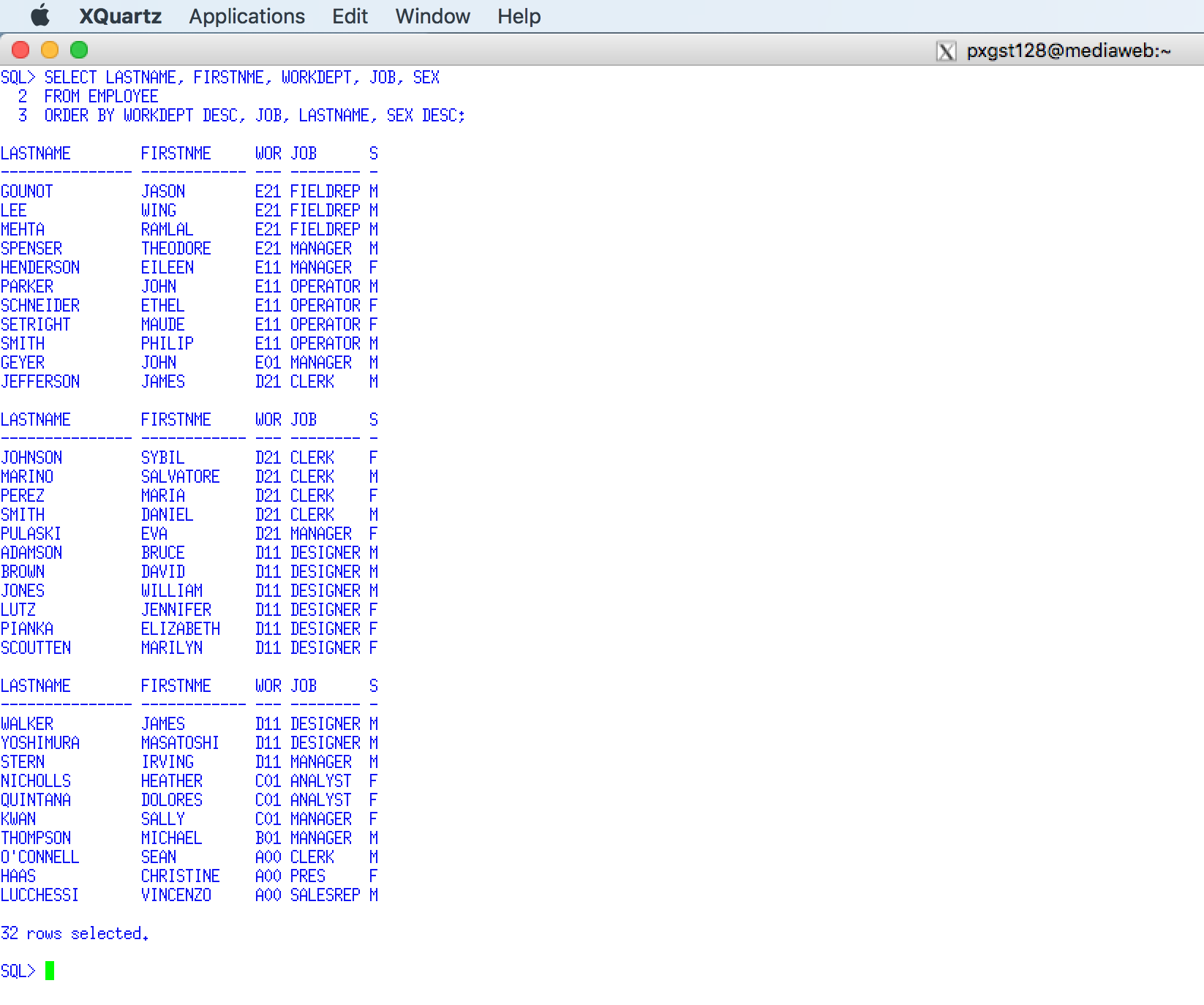
5. Another ORDER BY Example. Code the following SQL Statement followed by a semicolon to execute.



Use a Snippit to document your SQL statement =>



Use a Snippit to document your SQL output =>

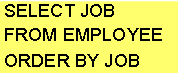


## 3.3 DISTINCT Clause

**Your snipping tool documentation must clearly display your Putty Banner.**

7. Suppressing Duplicate Output Rows - Requirement - **List the number of different jobs without duplicates**.

To demonstrate an example of a SELECT statement that list duplicate rows code the following SQL Statement followed by a semicolon to execute.



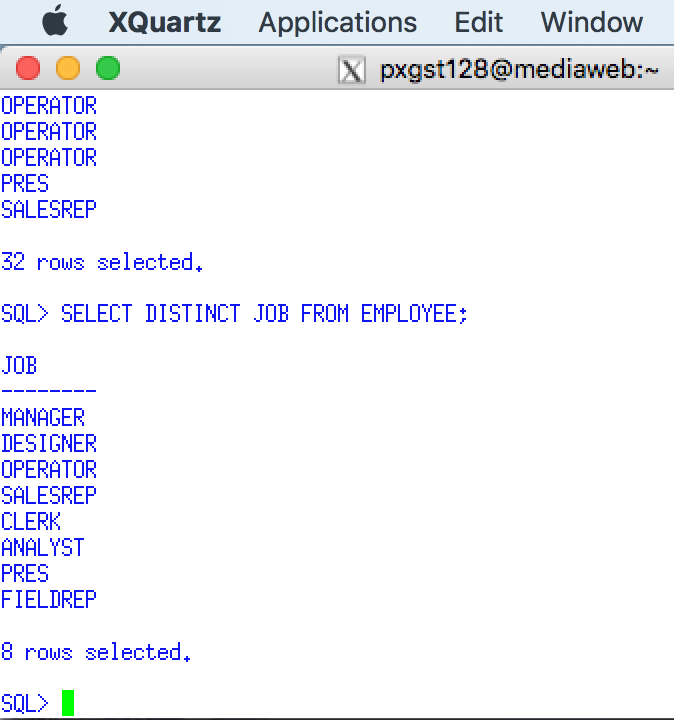
Use a Snippit to document your SQL output =>



To demonstrate an example of a SELECT statement that list duplicate rows code the following SQL Statement followed by a semicolon to execute.



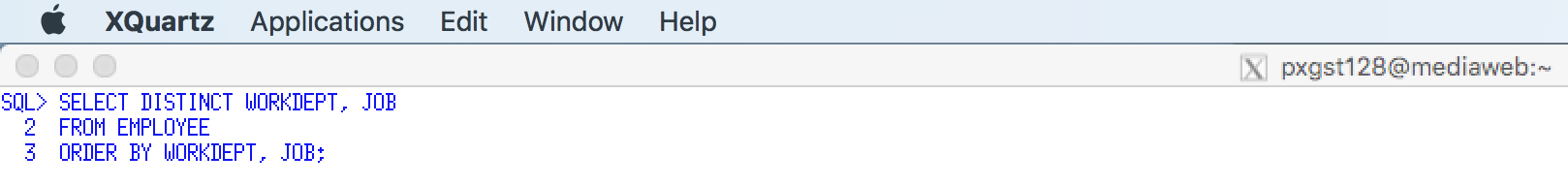
Use a Snippit to document your SQL output =>



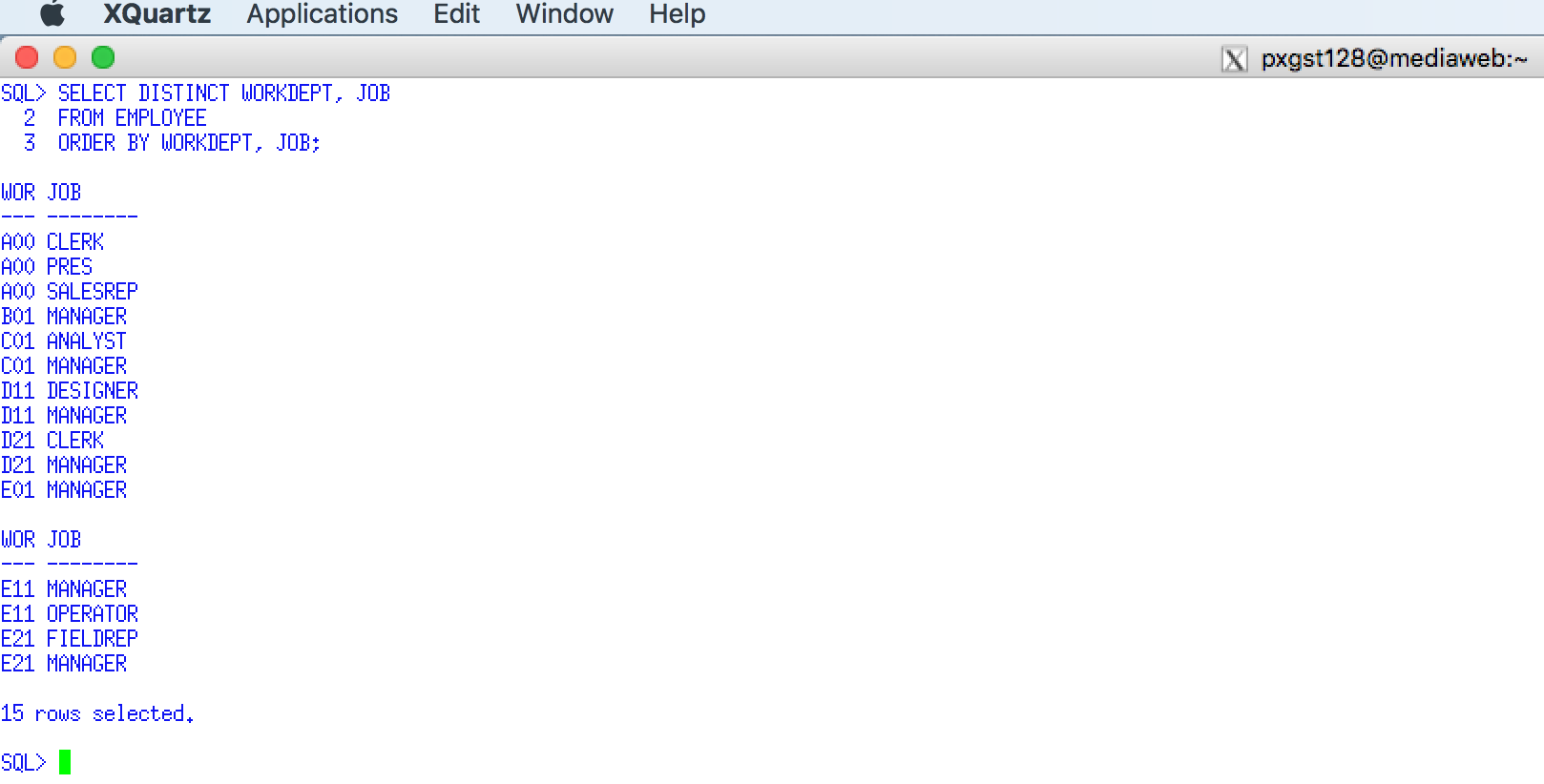
8. Suppressing Duplicate Output Rows - Requirement - **Provide a listing of the job distribution by department.** Code the following SQL Statement followed by a semicolon to execute.



Use a Snippit to document your SQL statement =>



Use a Snippit to document your SQL output =>

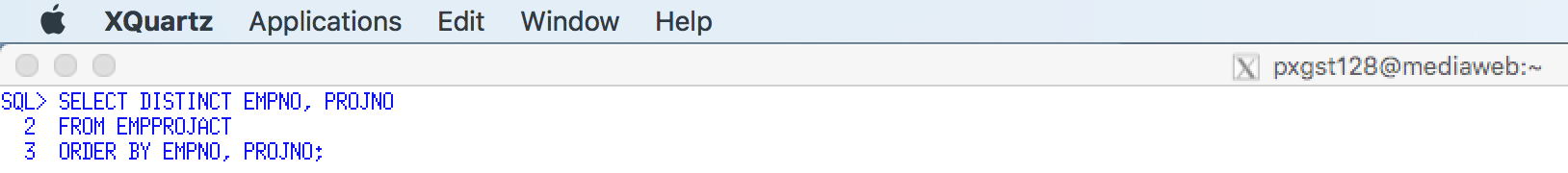


## 3.4 Your Requirement Example

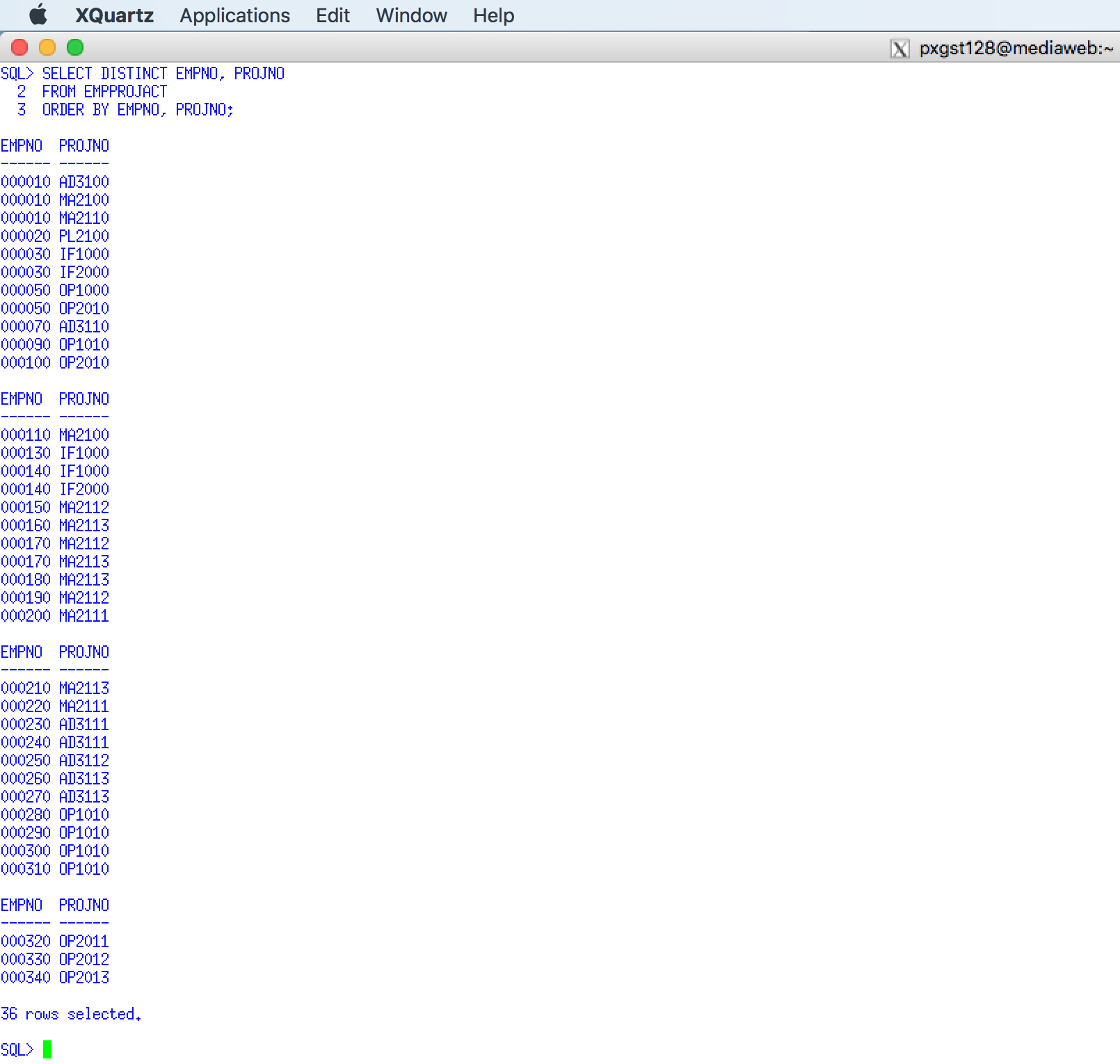
**Your snipping tool documentation must clearly display your Putty Banner.**

9. Your Requirement - Using EMPPROJACT table use the distinct clause to **Display the employee number and the project number they worked on, e.g., What projects did the employee charge their time to?** Since an employee may have many charges to a project you will need to use the DISTINCT clause. Sort by employee number. Code the previous SQL Statement requirements

Use a Snippit to document your SQL statement =>



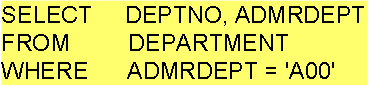
Use a Snippit to document your SQL output =>



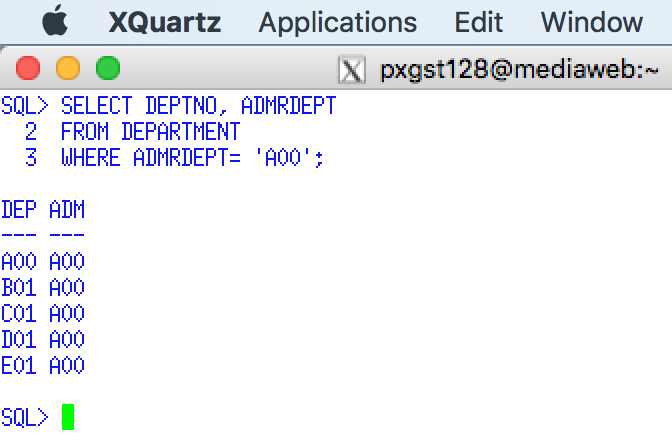
## 3.5 WHERE Clause

**Your snipping tool documentation must clearly display your Putty Banner.**

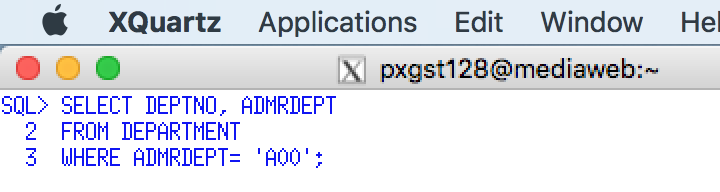
10. Retrieving Rows by Character Comparison - Requirement - **Provide a list of the departments reporting to department A00.** Code the following SQL Statement followed by a semicolon to execute.



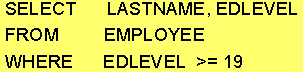
Execute this example and display the result. Use a Snippit to document your SQL output =>



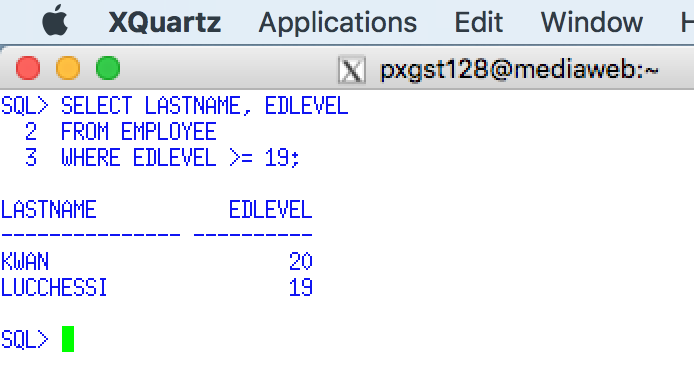
Use a Snippit to document your SQL statement =>



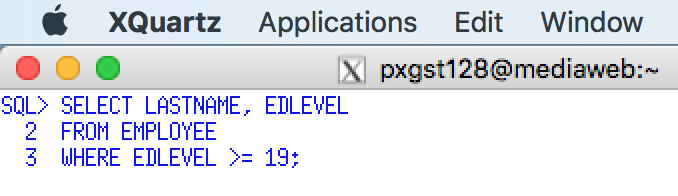
11. Retrieving Rows by Numerical Comparison - Requirement - **Using the employee table to display employee's lastname and education level above level 19.** Code the following SQL Statement followed by a semicolon to execute



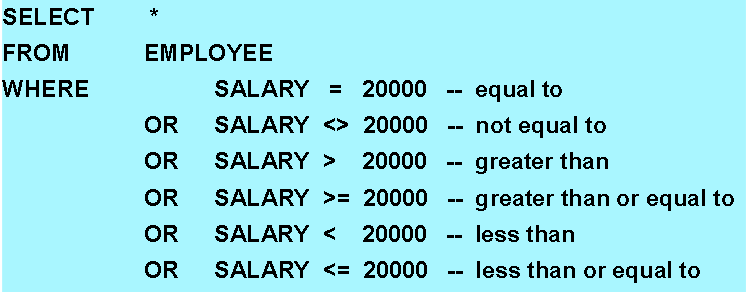
Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



## 3.6 Comparison Operators

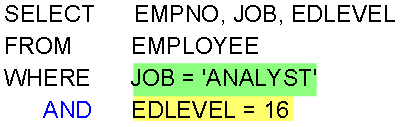


## 3.7 Logical Operators – AND, OR, NOT

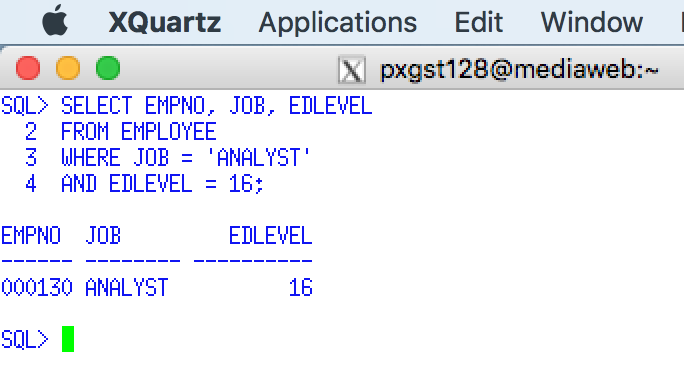
**Your snipping tool documentation must clearly display your Putty Banner.**

Multiple search conditions within a WHERE clause - http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Fsqlp%2Frbafymultiplewhere.htm

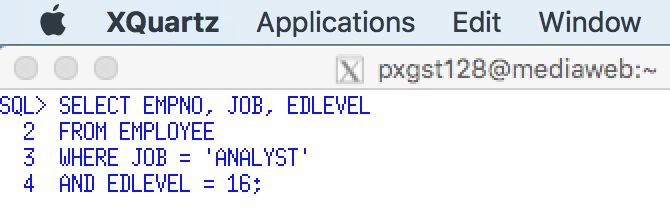
12. Multiple Conditions - AND operator - **Display employee's number, job and education level for the job position titled ANALYST and an education level of 16.**



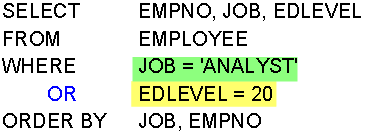
Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>

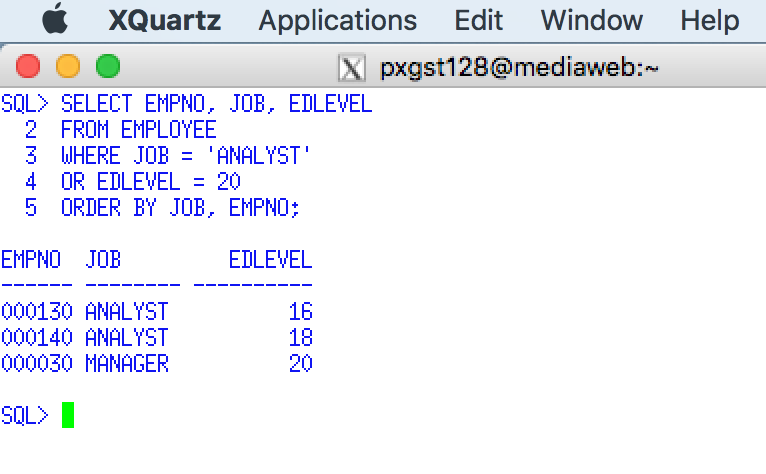


13. Multiple Conditions - OR operator - **Display the employee's number, job and education level for the job position titled ANALYST OR an education level of 20, then sorted job number and employee number.**

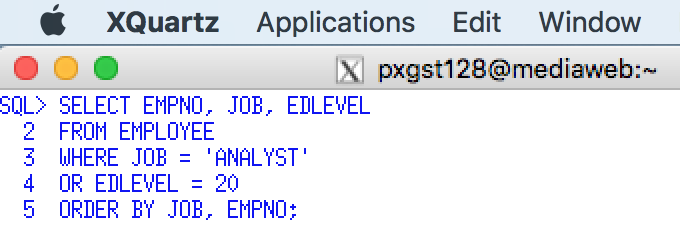


**Your snipping tool documentation must clearly display your Putty Banner.**

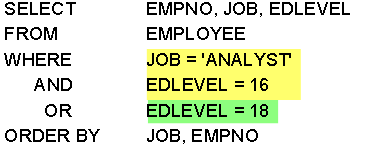
Execute this example and display the result. Use a Snippit to document your SQL output =>



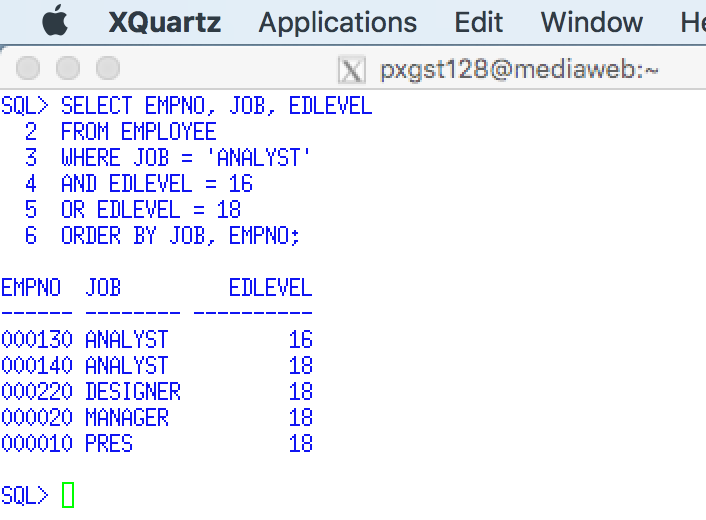
Use a Snippit to document your SQL statement =>



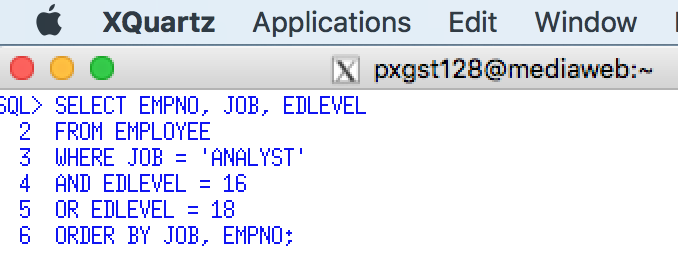
14. Multiple Conditions - AND/OR operator - **Display the employee's number, job and education level for the job position titled ANALYST with an education level of 16. Include all employees with an education level of 20, then sorted job number and employee number.**



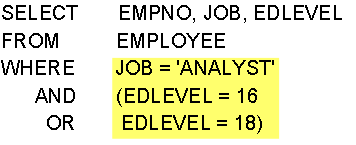
Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>

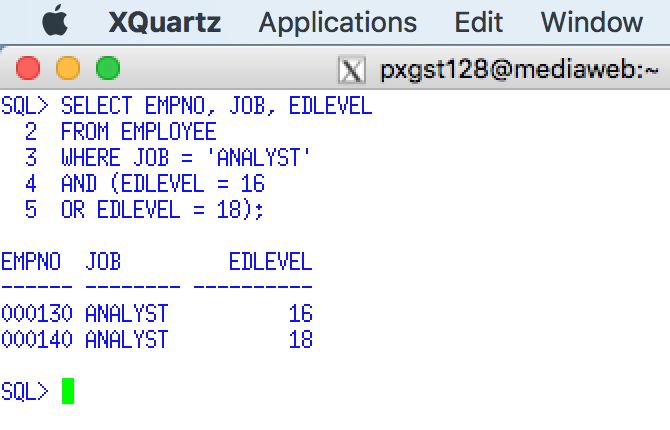


15. Multiple Conditions - AND/OR Parentheses - **Display the employee's number, job and education level for the job position titled ANALYST and an education level of 16 or 20.**

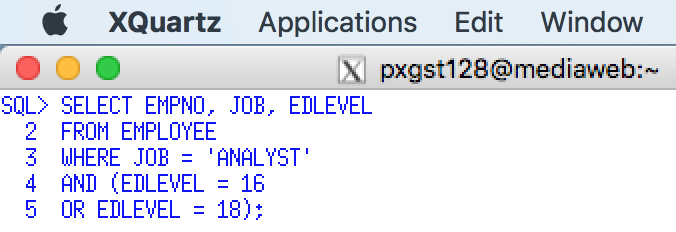


**Your snipping tool documentation must clearly display your Putty Banner.**

Execute this example and display the result. Use a Snippit to document your SQL output =>



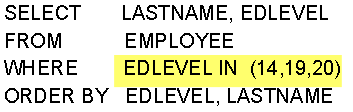
Use a Snippit to document your SQL statement =>



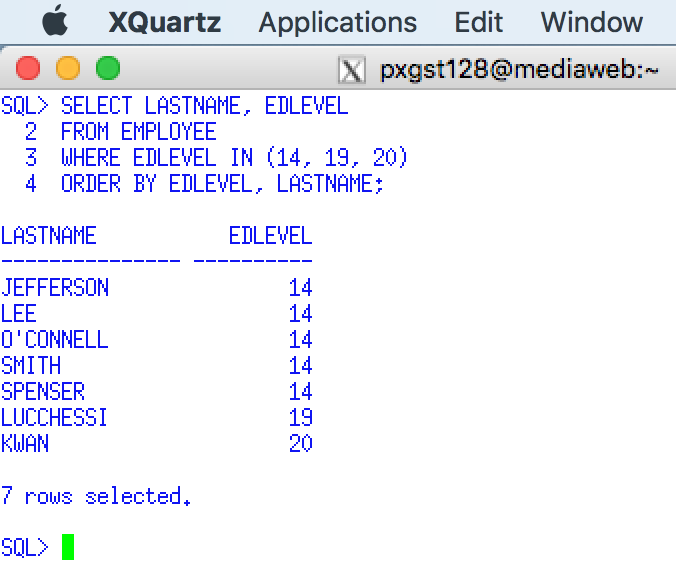
Special note. In the presentation the author demonstrate the use of AND and OR without the use of the parenthesis. This example is rare in real-world data retrieval. This requirement 15 is more realistic. Due to the order of logical comparisons, the AND operator is ALWAYS executed before the OR operator. Therefore, most requirements which use ANDs and ORs would require that the OR logical comparison be executed first, e.g., Does the employee first have an education level equal to 16 or 20, then compare the result with the AND logical operator, e.g., Is that education level a system analyst.

## 3.8 IN Operator

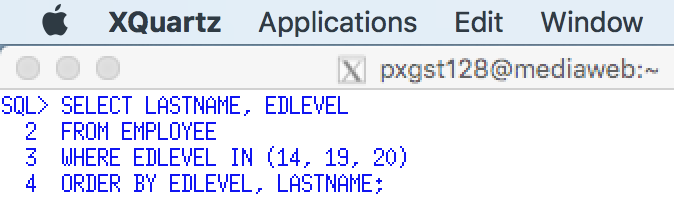
16. SELECT with IN - The following demonstrates the use of the WHERE clause with the IN operator using the employee table to **Display the employee's last name and education level with an education level of 14, 19 or 20, then sort by education level and last name..**



Execute this example and display the result. Use a Snippit to document your SQL output =>

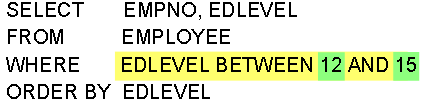


Use a Snippit to document your SQL statement =>



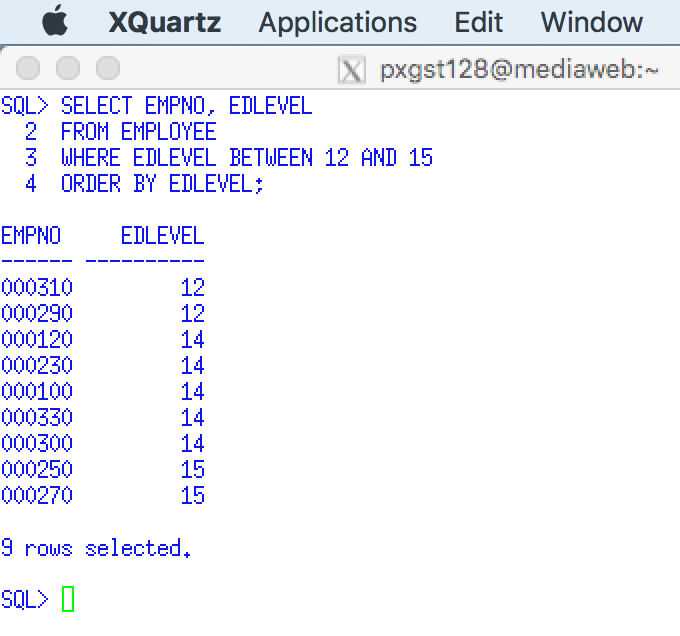
## 3.9 BETWEEN Operator

17. SELECT with BETWEEN - The following demonstrates the use of the WHERE clause with the BETWEEN operator using the employee table to **Display the employee's last name and education level with an education level between 12 and 15, then sorted by education level.**

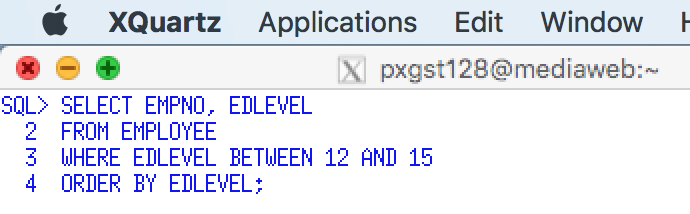


**Your snipping tool documentation must clearly display your Putty Banner.**

Execute this example and display the result. Use a Snippit to document your SQL output =>

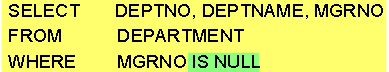


Use a Snippit to document your SQL statement =>

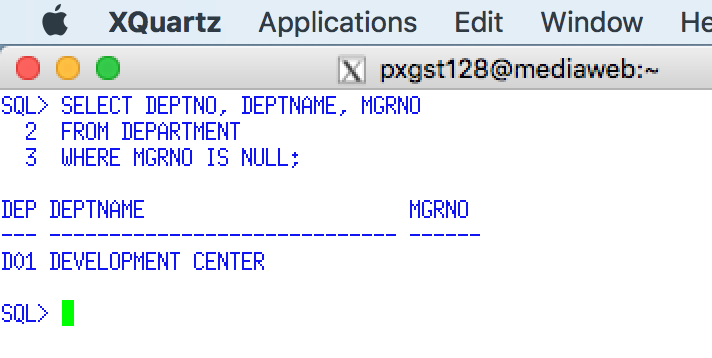


## 3.10 NULL Comparison

18. Retrieving Rows - Null Comparison - **Display the department no, department name and manager no from the department table where there is no manager assigned, e.g., the MGRNO is null.**



Execute this example and display the result. Use a Snippit to document your SQL output =>

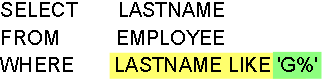


SQL: LIKE Condition - http://www.techonthenet.com/sql/like.php

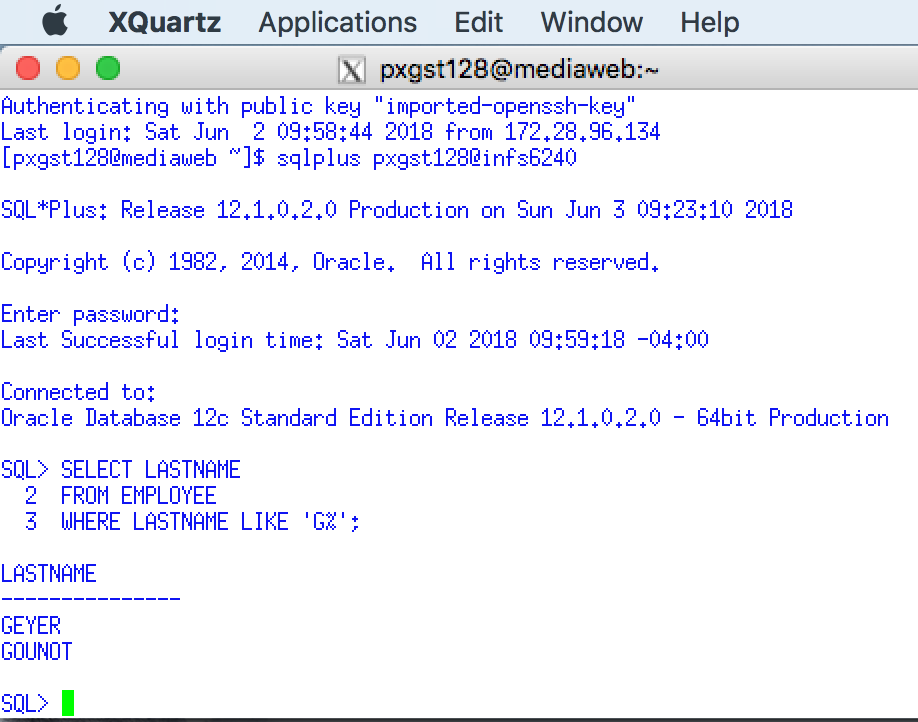
Special considerations for LIKE - <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Fsqlp%2Frbafymultiplewhere.htm>

LIKE predicate escape clause - <http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.odbc%2Fsrc%2Ftpc%2Fdb2z_likesc.htm>

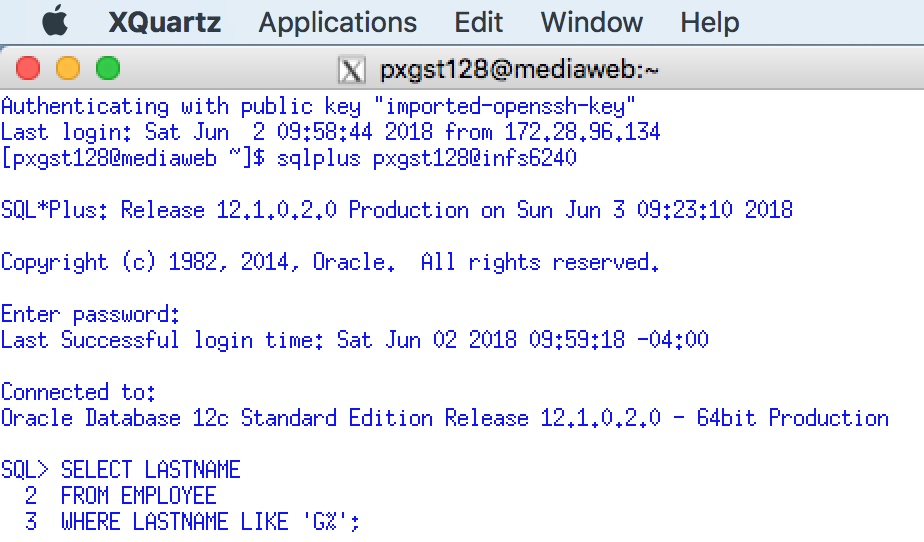
19. Partial String Search - LIKE '%' Comparison - **Display the employee last name for those employee's last name who begin with the letter "G".**



Execute this example and display the result. Use a Snippit to document your SQL output =>

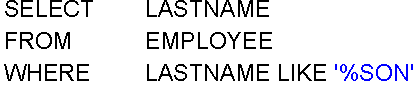


Use a Snippit to document your SQL statement =>



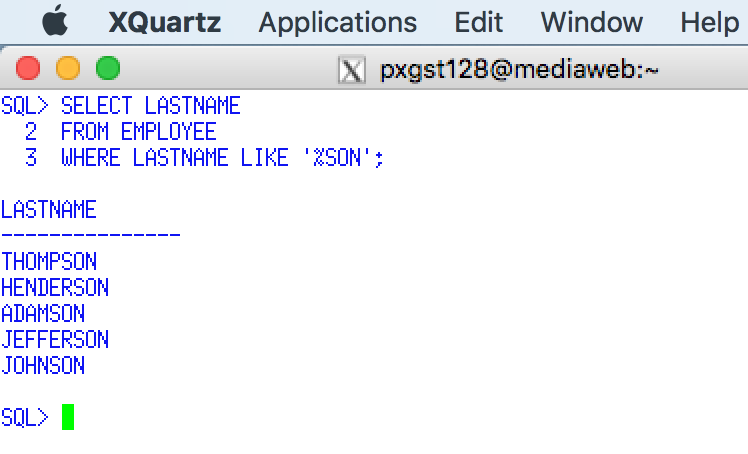
## 3.11 LIKE Operator – Partial String Match

20. Partial String Search - LIKE '%' Comparison - **Display the employee last name for those employee's last name who ends with "SON".**

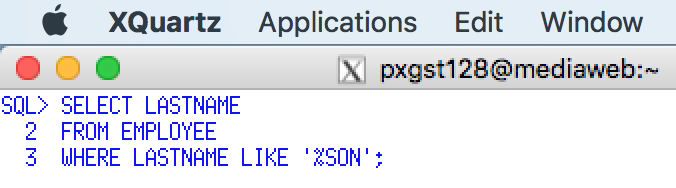


**Your snipping tool documentation must clearly display your Putty Banner.**

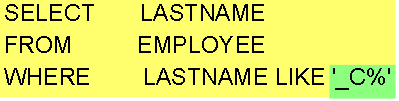
Execute this example and display the result. Use a Snippit to document your SQL output =>



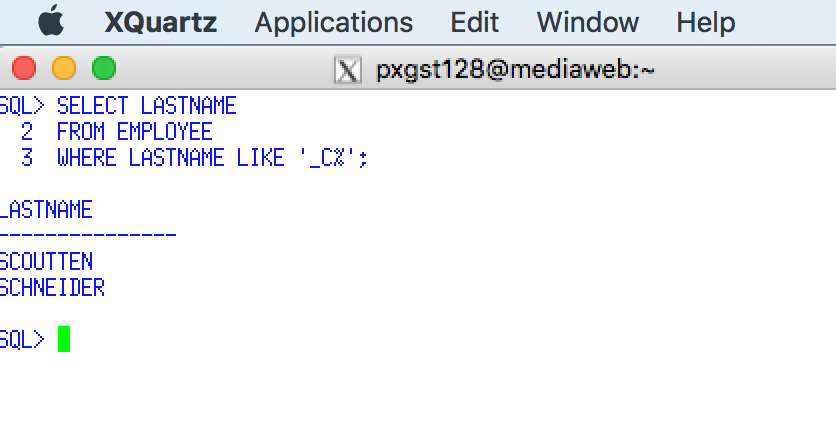
Use a Snippit to document your SQL statement =>



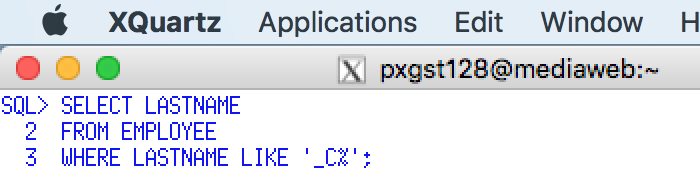
20. Partial String Search - LIKE '\_' Comparison. **Display the employee last name for those employee' last names who second character is the letter "C".**



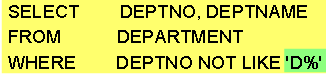
Execute this example and display the result. Use a Snippit to document your SQL output =>



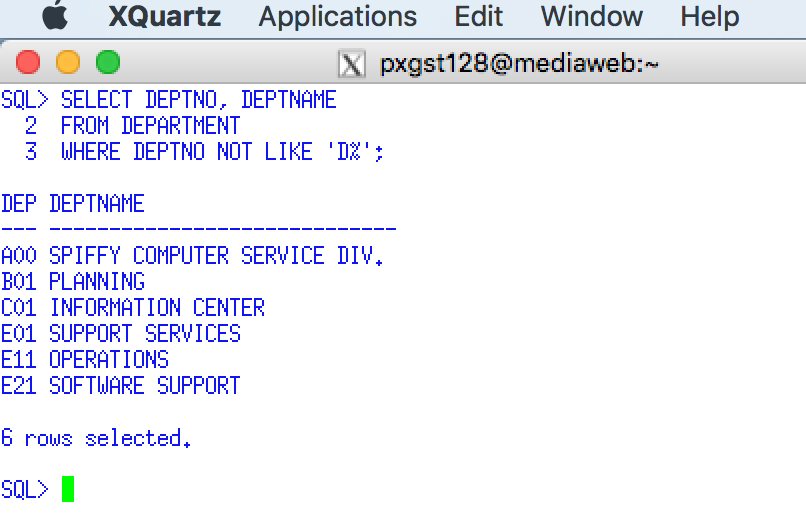
Use a Snippit to document your SQL statement =>



21. Negation - **Display ALL department numbers and department names, EXCEPT those who being with the letter "D".**



Execute this example and display the result. Use a Snippit to document your SQL output =>



## 3.12 Questions - SELECT review

1. True or False? The sequence in which clauses are coded in a SELECT statement is arbitrary. Answer =>

.trueSpring 2018

2. The BETWEEN clause causes the rows of the result table to be

a. Sorted in ascending order

b. Sorted in descending order

c. Not sorted at all

Answer =>

.a. sorted in ascending orderSpring 2018

3. Which clauses in a SELECT statement are required? Answer =>

.Spring 2018

4. Within a WHERE clause, character data, dates, times and timestamps must always be enclosed within which type of character? Answer =>

quote

.Spring 2018

5. Which SQL command can perform pattern matching using the underscore and the percent symbol? Answer =>

LIKESpring 2018

6, True or False? WHERE clause predicates must be separated from each other by commas. Answer =>

TRUE

.Spring 2018

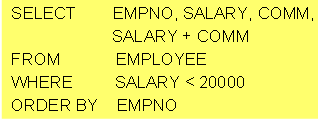
# Requirement 4. SQL Scalar Functions and Arithmetic

In this requirement you are required to individually execute **SQL> prompt** the sample SQL statements from this presentation and document the results. The following requirements will list the topic of individual slides. All SQL statements and output need to be documented using the Windows snippit tool or other comparable tool.

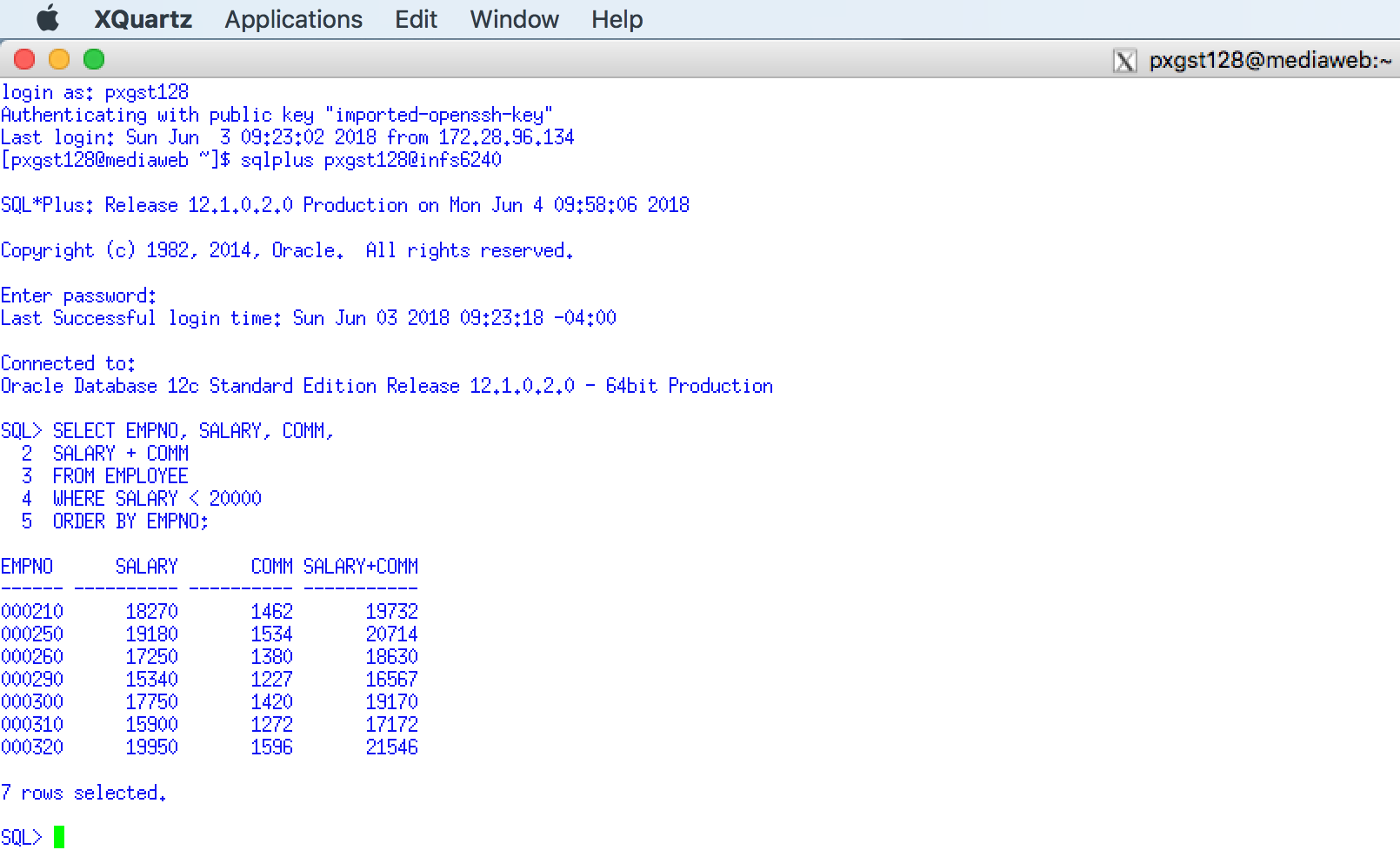
## 4.1 Calculated Columns

**Your snipping tool documentation must clearly display your Putty Banner.**

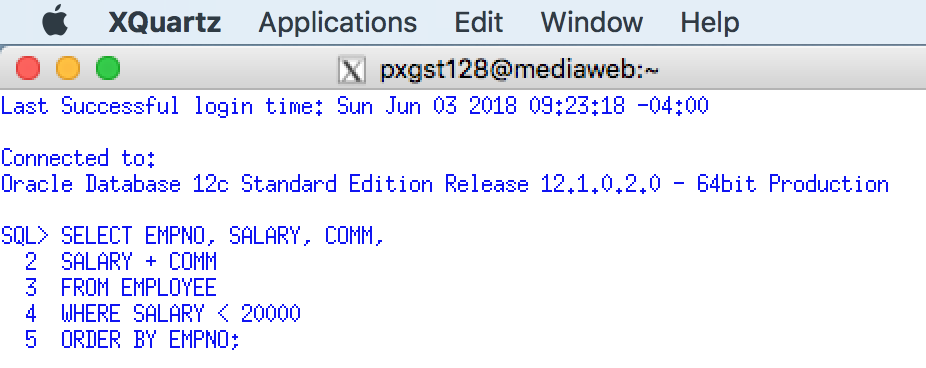
1. Selecting Calculated Values - **Display employee number, salary, commission, and a calculated column of salary + commission for those employees who have a salary less than $20,000, sorted by employee number.**



Execute this example and display the result. Use a Snippit to document your SQL output =>



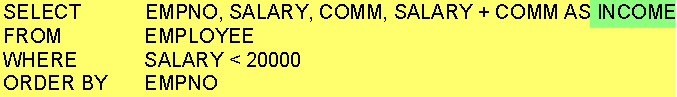
Use a Snippit to document your SQL statement =>



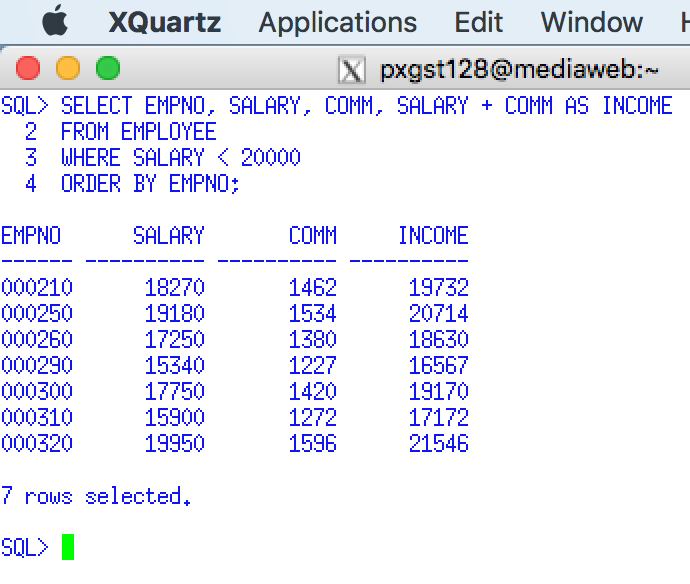
## 4.2 Naming Calculated Columns - AS

**Your snipping tool documentation must clearly display your Putty Banner.**

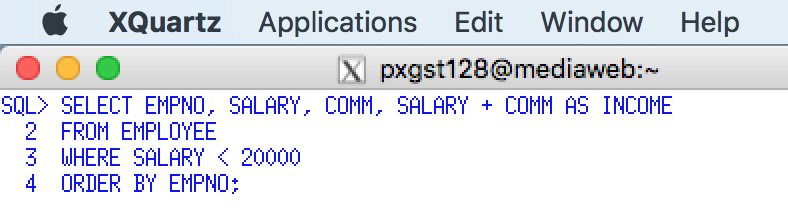
2. Naming Result Columns - The presentation demonstrates the use of the 'AS' clause to name a SQL displayed column. This SQL statement simply **re-names the calculated column as INCOME.**



Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>

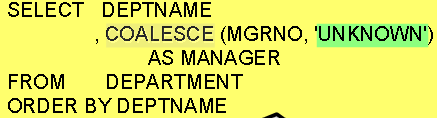
.Spring 2018

Special considerations: You may use the 'AS' column to name any column. If you do not think that the database column name is descriptive enough for your needs, you can declare your own column title. However the column name MUST not contain embedded spaces. At best you can try using the underscore character, e.g., ANNUAL\_INCOME. The format of the SELECT output is very limited. There are many report writers and other tools that can better format the SELECT output. SELECT is a data retrieval command, not a report writer.

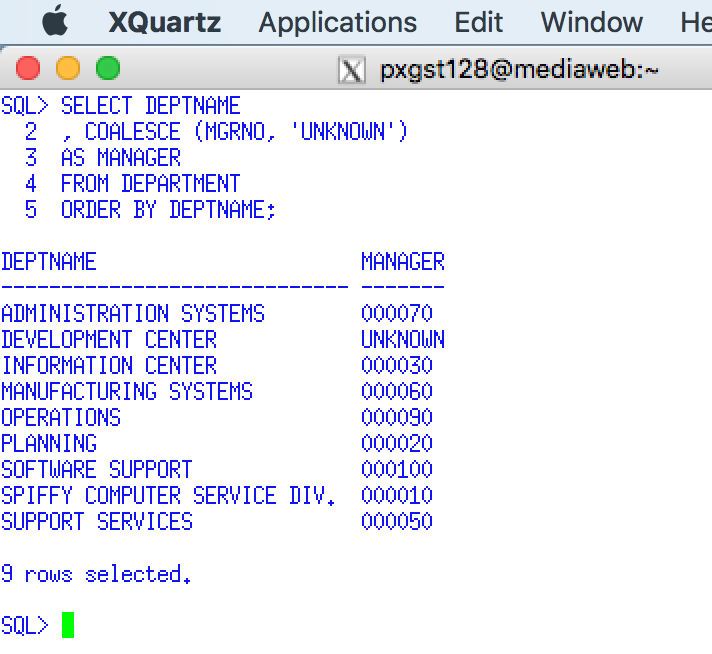
## 4.3 COALESCE Function

**Your snipping tool documentation must clearly display your Putty Banner.**

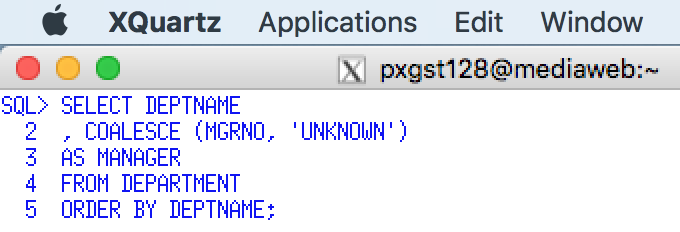
3. Substitution of NULL Values - COALESCE Function. The presentation demonstrates the use of the COALESCE function which display text for NULL values. **Display the department number and the manager number for each department. When a department has a NULL value for the manager number, you are display the message "UNKNOWN".**



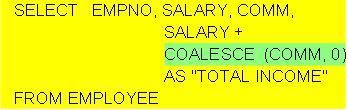
The manager number column should be titled MANAGER Execute this example and display the result. Use a Snippit to document your SQL output =>



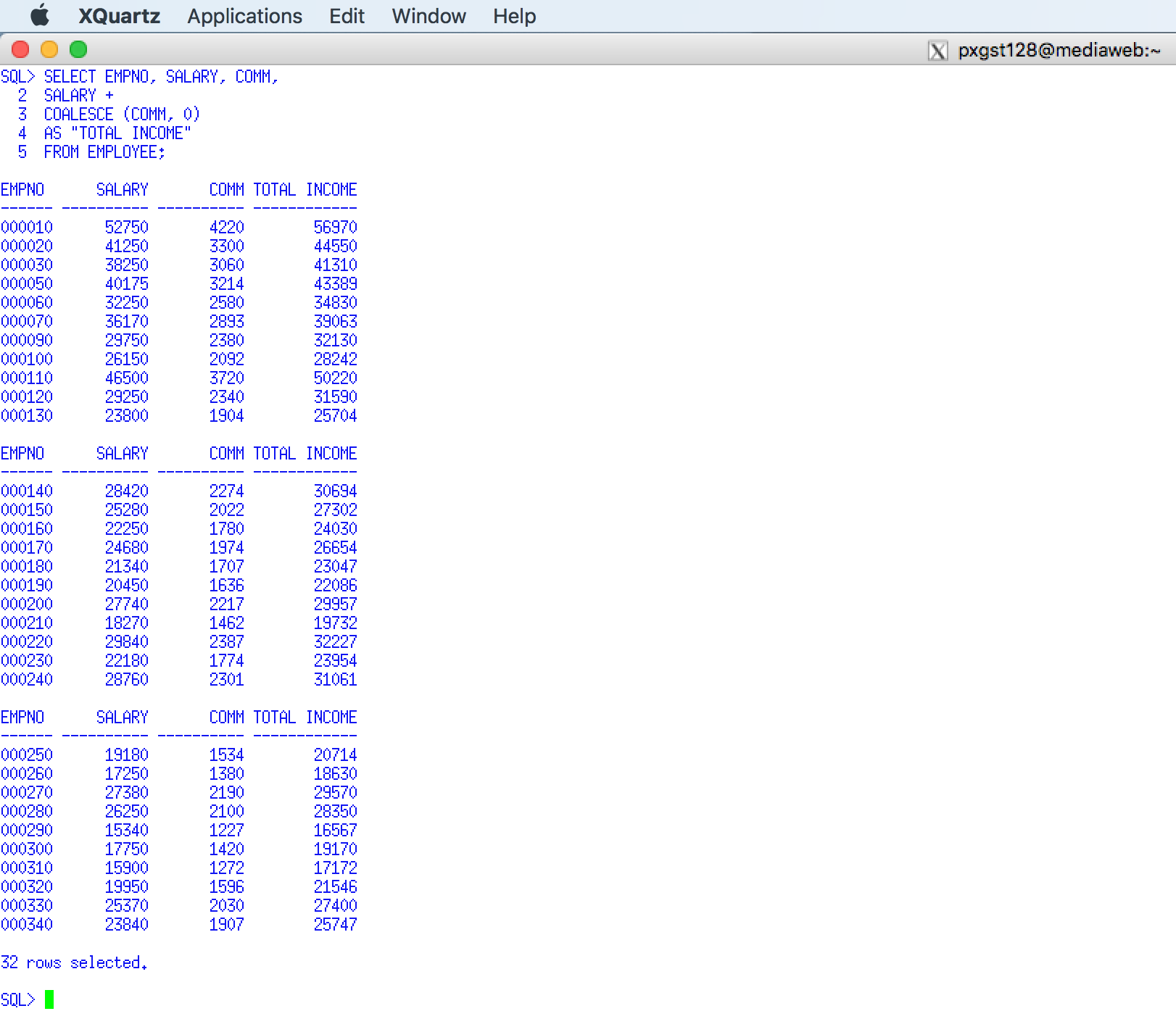
Use a Snippit to document your SQL statement =>



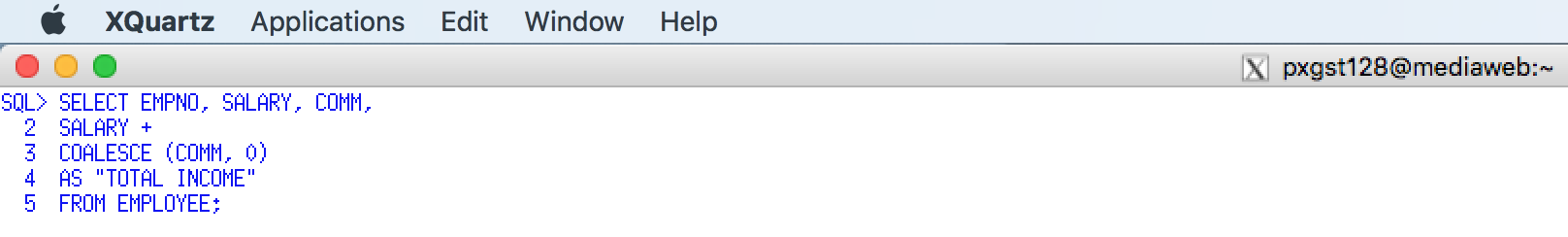
4. Arithmetic with NULL Values - COALESCE Function. The presentation demonstrates the use of the COALESCE function to substitute a zero for any NULL value. You may encounter problems involving any mathematical operation on NULL values. The COALESCE overcomes this problem. **Modify requirement 2 and use the COALSCE function to substitute a zero for all NULL commissions.** Select ALL employees.



Use a Snippit to document your SQL output =>

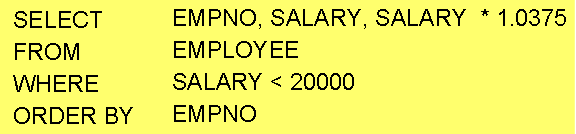


Use a Snippit to document your SQL statement =>

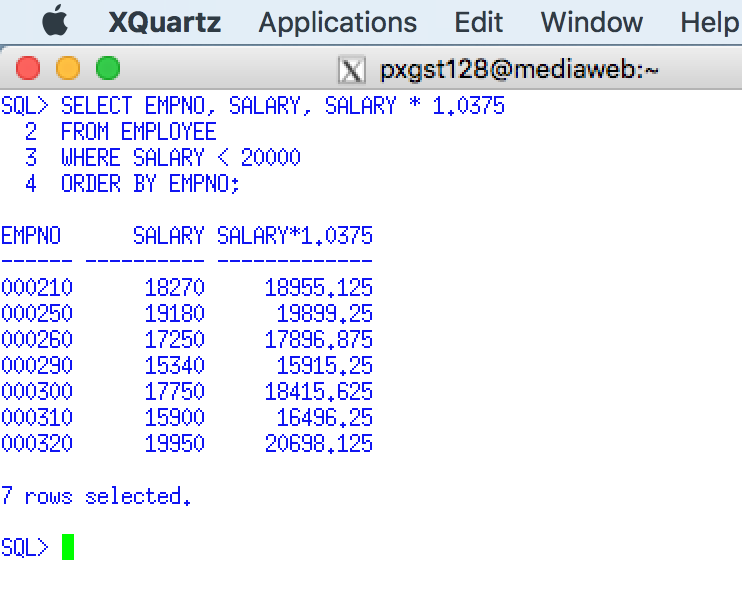


.Spring 2018

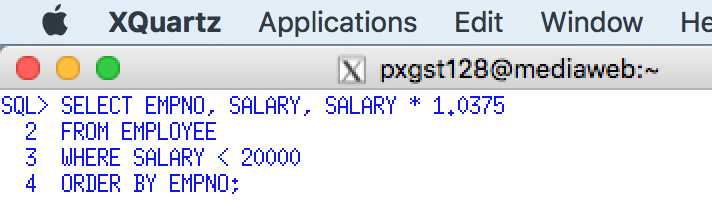
5. Calculated Values- **Display the employee number, current salary (the one stored in the employee table) and a calculate salary (the current salary with a 3.75% raise). Display only salaries less than $20,000.**



Execute this example and display the result. Use a Snippit to document your SQL output =>



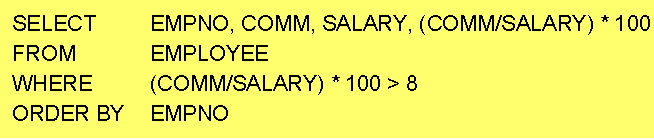
Use a Snippit to document your SQL statement =>



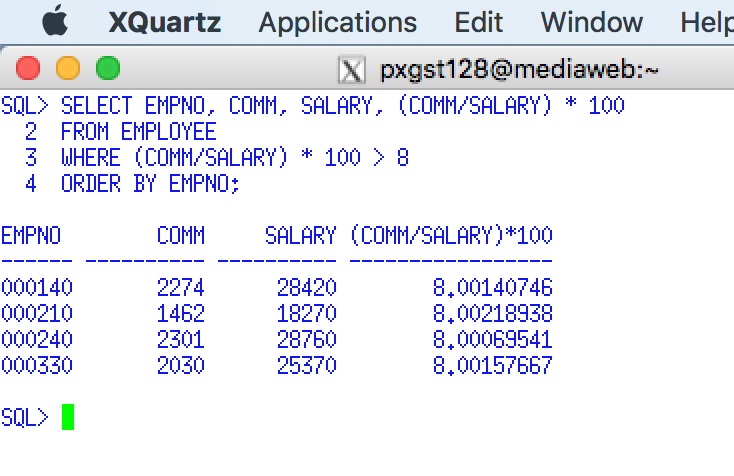
## 4.4 Calculate Values with the WHERE Clause

**Your snipping tool documentation must clearly display your Putty Banner.**

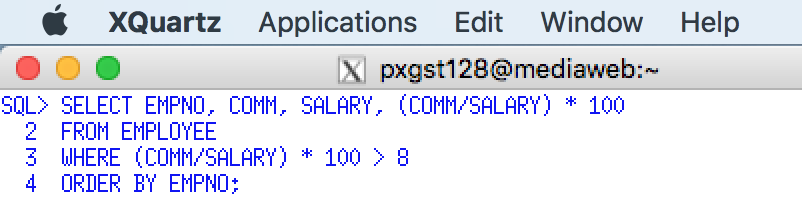
6. Condition on Calculated Values - **Display the employee number, commission, and the percentage of commission to salary (COMM/SALARY) \* 100 for those employees who make more than 8% commission.**



Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



## 4.5 SUBSTR Function

**Your snipping tool documentation must clearly display your Putty Banner.**

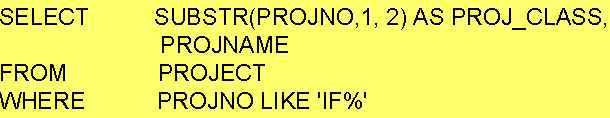
Designed for strings

SUBSTR scalar function - <http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.admin.doc%2Fdoc%2Fr0000785.htm>

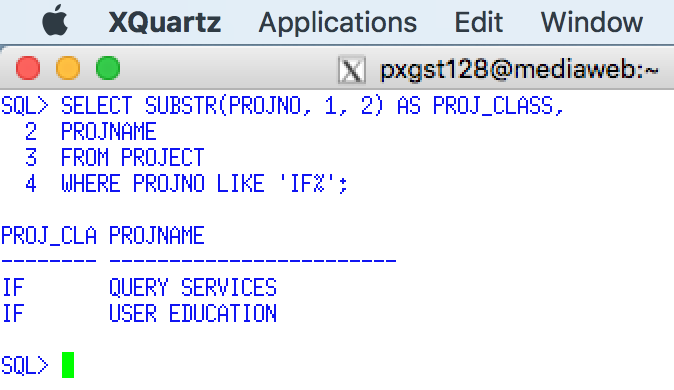
SUBSTRING scalar function - <http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.admin.doc%2Fdoc%2Fr0000785.htm>

POSITION scalar function CHAR data types only - <http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.admin.doc%2Fdoc%2Fr0000785.htm>

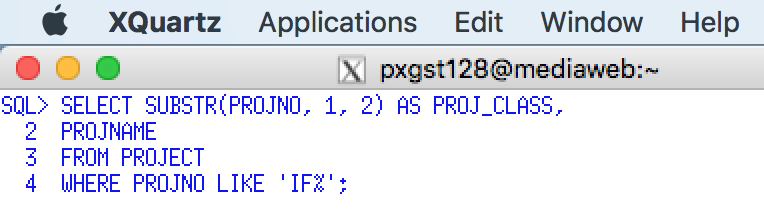
7. SUBSTR function - The presentation demonstrates the use of the SUBSTR function to select a portion of a string. For example, SUBSTR(LASTNME,3,2) returns the characters beginning at position 3 and a length of two characters of the last name. If the last name was 'SMITH' then the characters 'IT' would be the result.



The example would **Display the first two characters of the project number and project name for any project number starting with 'IF.** Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



## 4.6 Concatenation

CONCAT scalar function - http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.admin.doc%2Fdoc%2Fr0000785.htm

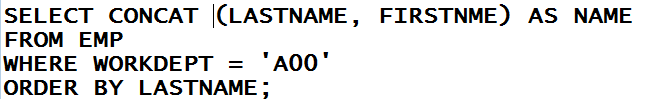
14. Concatenation of values - the CONCAT scalar function of || - DB2 is concatenation to combine two or more character columns or strings to display a new column. You cannot concatenate numeric data types or constants without converting the data to a string, e.g., use the CHAR function.

The presentation provides an example to **Display the employee's last name with a comma and the first name** e.g., Smith, John.

The Oracle SQL CONCAT function is different that the DB2 SQL CONCAT function.

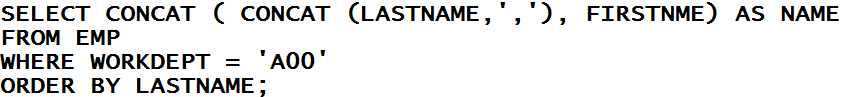
Every SQL function must use a set of parenthesis and contains input arguments. These work like EXCEL functions. Most of are familiar with the EXCEL SUM function, e.g., =SUM(A1:C1). Inside the SUM(inputs) that are to be summed or totaled, or the range of cells between A1 and C1.

The CONCAT function contains two input arguments. Give the example, CONCAT('GO' , ' STEELERS') . Two literal includes in a single quote that will be combined in create the phrase GO STEELERS. One could use the CONCAT function to combine the data contents of two columns, e.g., CONCAT(LASTNAME, FIRSTNME). The result would be HASSCHRISTINE. Notice the content of the two columns will be concatenated (combined) without a space between the lastname and the firstnme. The following is the SQL statement will work, but would be visually poor.

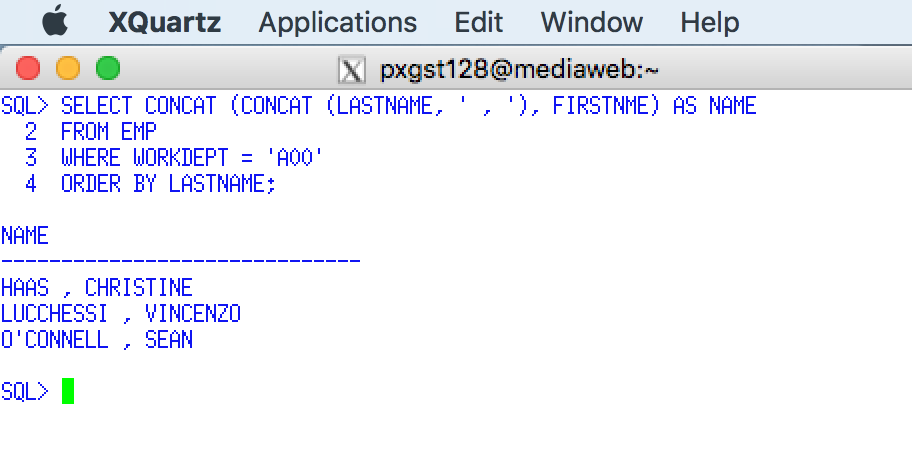


It would be preferred to place a command and an extra space between the LASTNAME and the FIRSTNME to make the combination more visually appealing, e.g., HASS, CHRISTINE. The trouble is that the CONCAT function can only combine two strings at a time. The following example will code a CONCAT function within a second CONCAT function. To remind you, all functions require two parentheses, i.e., a left and a right parenthesis. If can you count an odd number of parentheses you have a problem.

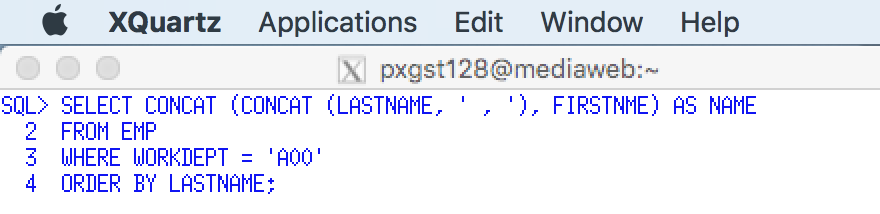
The following example means 1) combine the LASTNAME with a comma and a space, 2) then combine the FIRSTNAME with the previous result.



Execute this example and display the result. Use a Snippit to document your SQL output =>

.Spring 2018

Use a Snippit to document your SQL statement =>



Since the CONCAT function may be difficult to code you can also use the || operator.

**SELECT LASTNAME || ', ' || FIRSTNME AS NAME**

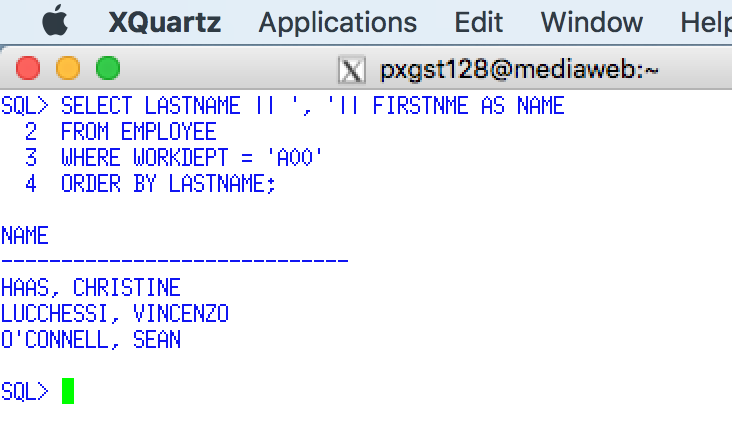
**FROM EMPLOYEE**

**WHERE WORKDEPT = 'A00'**

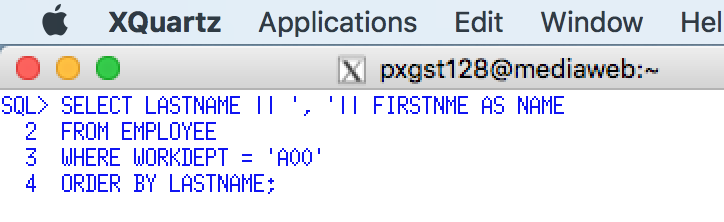
**ORDER BY LASTNAME**

It is common that a person's name be stored in three or more columns, e.g., first name, middle initial, and last name, or perhaps a title or suffix. Breaking a person's name provides more processing flexibility. But that processing flexibility means that we need to learn how to sue concatenation to combine the separate columns or literals to display a more user-friendly string.

Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



# Requirement 5. Column Functions and Grouping

In this requirement you are required to individually execute **SQL> prompt** the sample SQL statements from this presentation and document the results. The following requirements will list the topic of individual slides. All SQL statements and output need to be documented using the Windows snippit tool or other comparable tool.

Scalar functions - <http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.sqlref%2Fsrc%2Ftpc%2Fdb2z_scalarfunctionsintro.htm>

Aggregate functions - <http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.sqlref%2Fsrc%2Ftpc%2Fdb2z_scalarfunctionsintro.htm>

SQL Functions - <http://www.bin-co.com/database/sql_tutorial/sql_functions.php>

http://www.1keydata.com/sql/sql-as.html

Select Statement and Where Clause

http://www.youtube.com/watch?v=fPFHjt4pVas&feature=fvwrel

Count Function -

http://www.youtube.com/watch?v=6qxvXvLIojo&feature=fvwrel

Sum Function -

http://www.youtube.com/watch?v=ZOh4u-aqLC8&feature=fvwrel

Sum and Coalesce Functions -

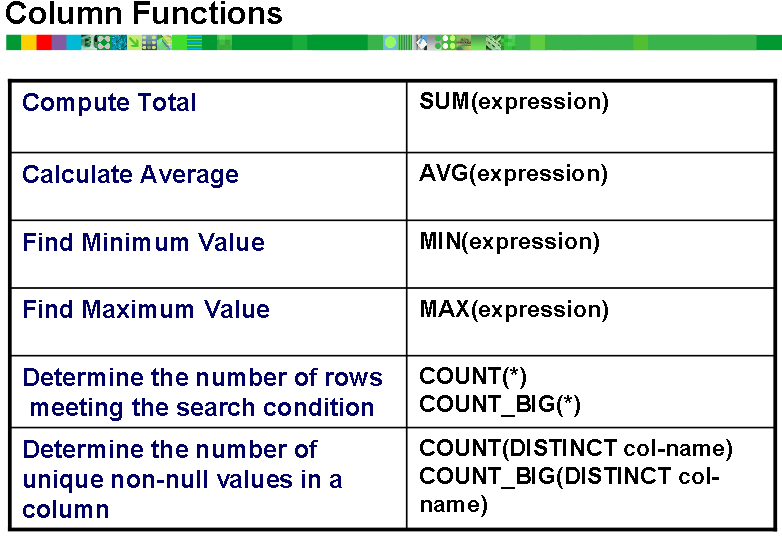
http://www.youtube.com/watch?v=trDKWEpVXnI&feature=fvwrel

Null Values -

http://www.youtube.com/watch?v=1ZrqQy-W8kA&feature=relmfu

Min, Max and AVG Functions

http://www.youtube.com/watch?v=5Bkdc-hLj9w&feature=relmfu



An objective is to clarify the difference between a SQL Scalar Function and a SQL Column Function. Most versions of SQL never use the term Column Functions. For the following requirement a Column Function will be referred to as an Aggregate Function. This is more appropriate.

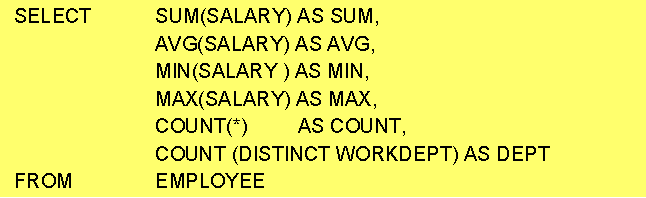
## 5.1 Basic Column Functions

1. What is the difference between an SQL Scalar Function, e.g., DATE(), YEAR(), TRIM(), etc., and an Aggregate Function, e.g., AVG(), MAX(), MIN(), etc.? Answer =>

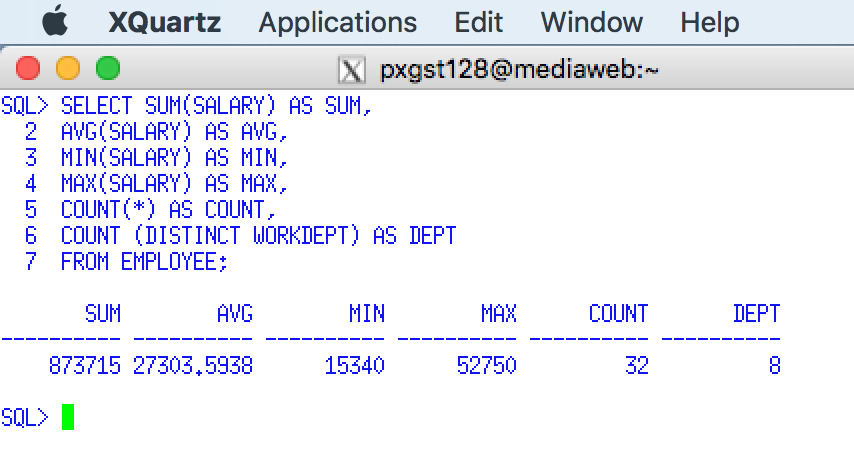
SQL Scalar function works on single data values, and an aggregate function works on the entire set of data ,not on one data element.

2. Sample Aggregate (Column) Functions - The presentation demonstrates the use of popular SQL aggregate functions, SUM(), AVG(), MIN(), MAX(), and COUNT() on the column SALARY in the employee table

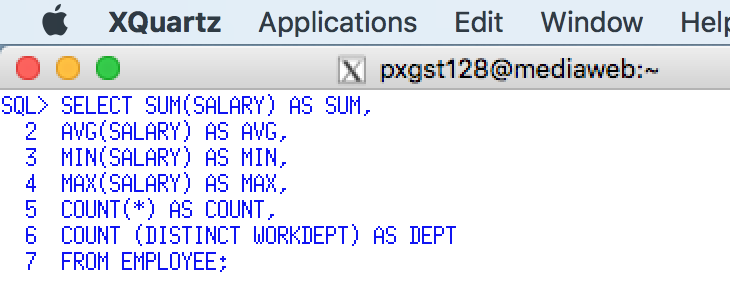
.



Execute this example and display the result. Use a Snippit to document your SQL output =>



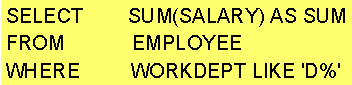
Use a Snippit to document your SQL statement =>



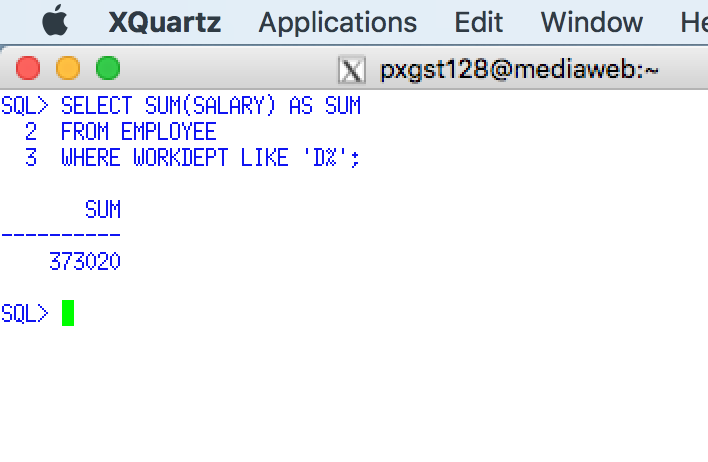
Special considerations: Some presentations display a mini-report which includes details for each employee and then a total line, which demonstrates aggregate function at the end of a detailed report.

You cannot code a combination detail listing and aggregate totals with one SQL statement. Either one codes a one SQL statement to generate detailed information or aggregate SQL to generate totals - you cannot do both. For example, SELECT EMPNO, SALARY, BONUS, COMM, SUM(SALARY), SUM(BONUS), SUM(COMM) … is illegal.

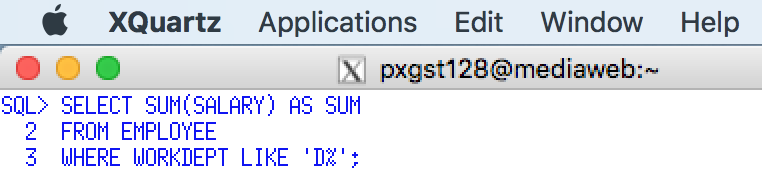
3. Aggregate (Column) Functions based on a Subset (using the WHERE clause). **Display the sum of all salaries for workdepts beginning with the letter D.**



Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>

.Spring 2018

DB2 Group By and Order By Clauses - <http://www.techtricky.com/db2-group-by-and-order-by-clauses/>

GROUP BY and HAVING clauses, Scalar Functions http://www.mainframegurukul.com/srcsinc/database/db2\_tutorials/sql\_tutorials/DB2\_SQL\_TUTORIAL\_5.html

DB2 Group By-clause - http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.sqlref%2Fsrc%2Ftpc%2Fdb2z\_sql\_groupbyclause.htm

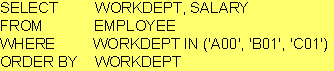
SQL Group by - <http://www.1keydata.com/sql/sqlgroupby.html>

DB2 Having clause <http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.sqlref%2Fsrc%2Ftpc%2Fdb2z_sql_groupbyclause.htm>

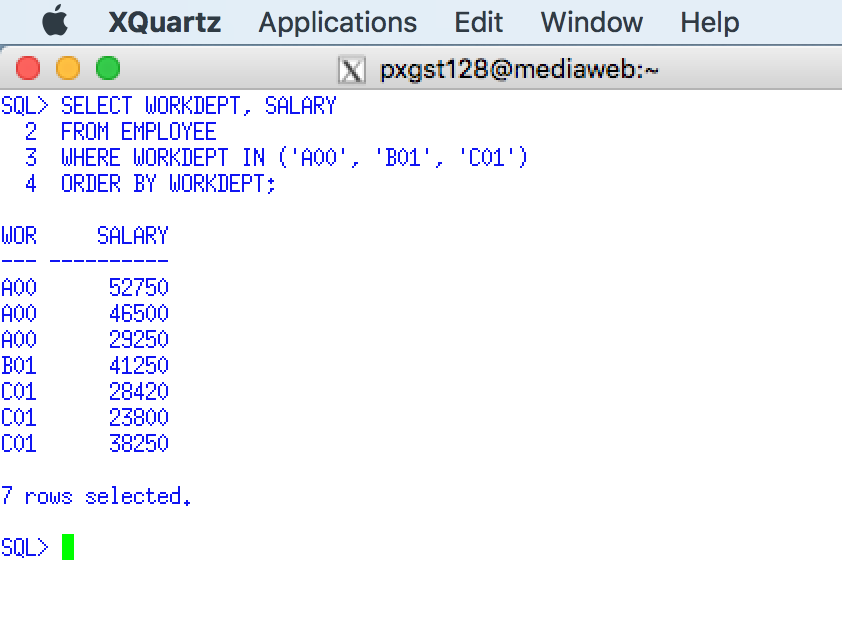
SQL: Group By, Having, Min, Max Sum - <http://www.youtube.com/watch?v=E3rXVrTTROU>

SQL GROUP BY, HAVING and aggregate functions, video tutorial - <http://www.youtube.com/watch?v=098hXV0q9ps>

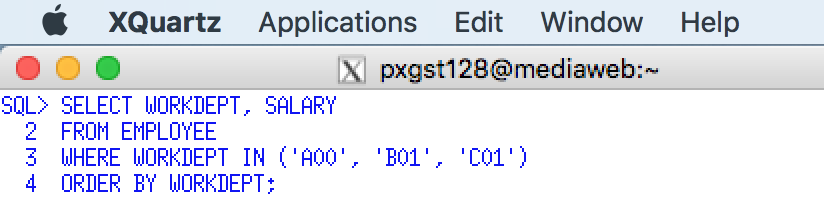
Execute the following SQL Statement.



Use a Snippit to document your SQL output =>

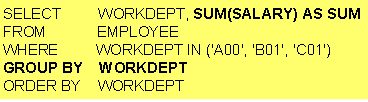


Use a Snippit to document your SQL statement =>

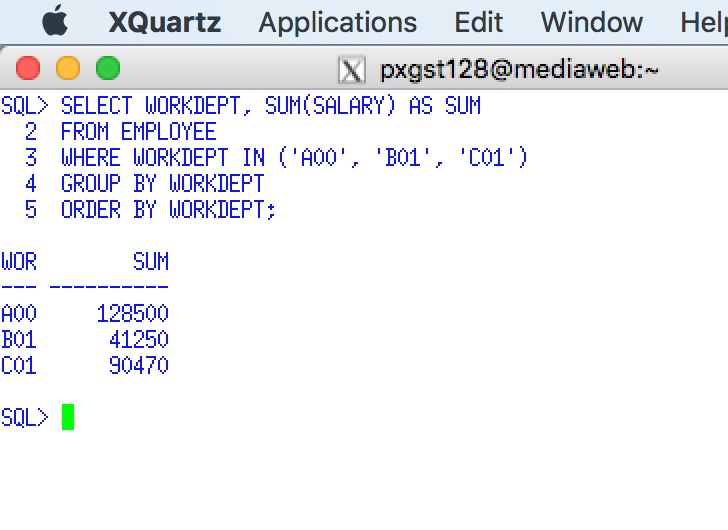


## 5.2 GROUP BY

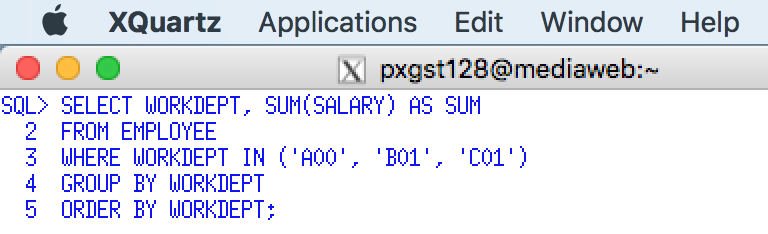
4. GROUP using one column. **Display the total of all salaries and work department number for departments A001, B001, and C001.**



Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



5. GROUP BY for more than ONE Column - Using the example provided in the presentation **Display the average salary per education level for each department group (given by the first character of the department number), for education levels 18 and higher**.

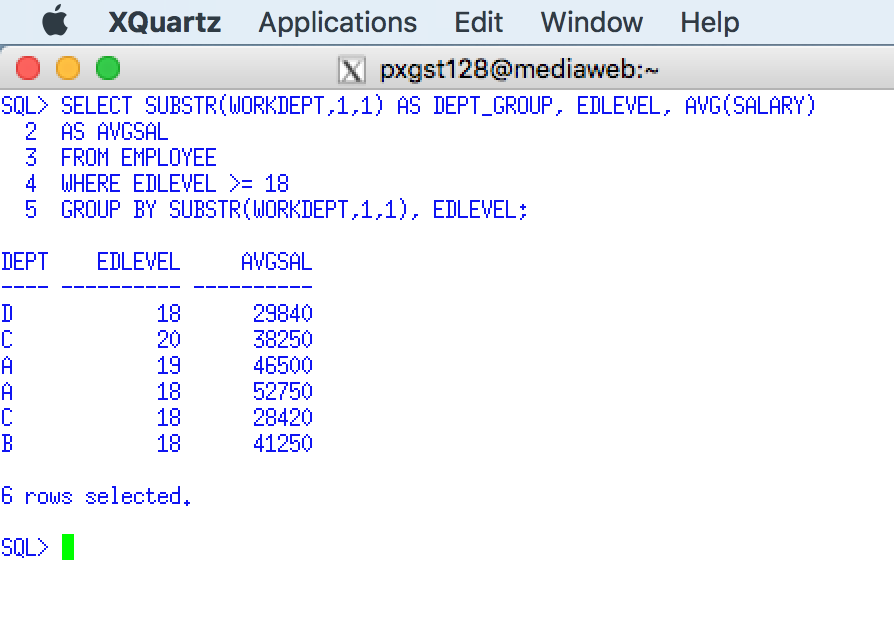
**SELECT SUBSTR(WORKDEPT,1,1) AS DEPT\_GROUP, EDLEVEL, AVG(SALARY) AS AVGSAL**

**FROM EMPLOYEE**

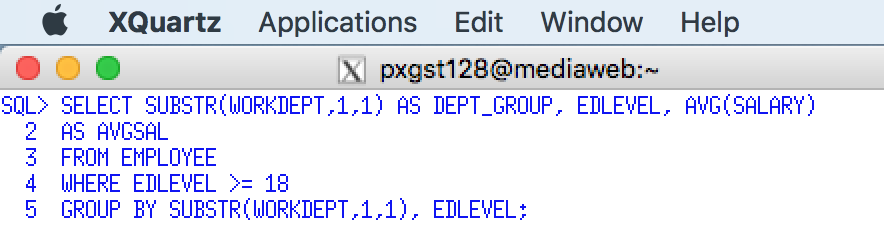
**WHERE EDLEVEL >= 18**

**GROUP BY SUBSTR(WORKDEPT,1,1), EDLEVEL**

Execute this example and display the result. Use a Snippit to document your SQL output =>



Use a Snippit to document your SQL statement =>



6. The previous SQL example contained the following function: SUBSTR(WORKDEPT,1,1). Why was the SUBSTR Function used? Answer =>

7. The previous SQL example contained the following function: SUBSTR(WORKDEPT,1,1). Is the SUBSTR function an Aggregate (COLUMN) function or a Scalar function? Answer =>

## 5.3 Incorrect GROUP BY Statements

11. Given the following SQL statements illustrating popular errors in the use of the GROUP BY clause and Aggregate (Column) Functions. Using the following table CLEARLY describe each GROUP BY or Aggregate Function error.

|  |  |
| --- | --- |
| **Incorrect GROUP BY or Aggregate Function Error** | **What is wrong with these GROUP By SQL statements?** |
| **SELECT EMPNO, WORKDEPT, SUM(SALARY)**  **FROM EMP**  **GROUP BY WORKDEPT** |  |
| **SELECT EMPNO, AVG(HIREDATE)**  **FROM EMP**  **GROUP BY EMPNO** |  |
| **SELECT EMPNO, WORKDEPT, SUM(SALARY)**  **FROM EMP**  **GROUP BY WORKDEPT** |  |

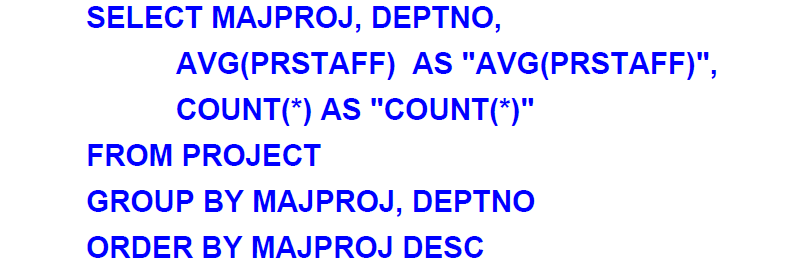
In the Project table there are two related columns, MAJPROJ and PROJNO. Both columns may contain a Project Number. A Major Project Number may contains sub-projects. For example, when the MAJPRJOG column is NULL, it is the PROJNO column and the row values are a Major Project, e.g., PROJNO AD3100. On the other hand, when the MAJPROJ column is not NULL, e.g., it contains the value AD3100, the PROJNO column and the row values are a Sub-Project, e.g., PROJNO AD3110. Therefore MAJPROJ column may contain duplicate values since the Major Project may have many subproject and departments. However, the PROJ Number uniquely identifies one project. (See sample data at the end of the assignment)

In the following requirement, the SQL statement begins with SELECT MAJPROJ, DEPTNO, … The objective of the SELECT is to analyze the departments (DEPTNO) within a major project (MAJPROJ). The average staffing (PRSTAFF) and the number of subprojects with the DEPTNO are analyzed. In the sample printout, review MAJPROJ AD3100 and MA2100.

* AD3100 uses resources from only one department D21 (there is only one row). MA2100 uses resources from only two departments: B01 and D11 (there are two rows).
* AD3100 has three subprojects which uses resources from department D21, e.g., the count is 3. MA2100 has one subproject which uses resources from department B01, e.g., the count is 1, and also has one subproject which uses the resources from department D11, e,g., the count is 1.

## 5.4 GROUP BY, ORDER BY

12. GROUP BY, ORDER BY - Multiple Columns -



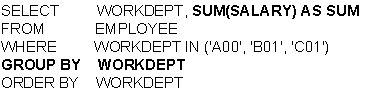
The previous SQL statement presents and analysis of present average staffing by subprojects/departments and the number of subprojects used for each major department, listed in descending order by major project number . Execute this example and display the result. Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

13. Given the previous requirement the display of the AVG(PRSTAFF) is poor. Which function would you use to make the AVG(PRSTAFF) column more readable? Answer =>

## 5.5 GROUP BY, WHERE, and HAVING

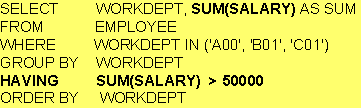
14. GROUP BY, HAVING (1 of 2) -



Execute this example and display the result. Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

**List the departments with total spent for salaries higher than $50,000**



Execute this example and display the result. Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

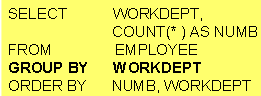
15. Display the jobs within a work department, excluding managers, designer, and field representative, with an average salary higher than $25,000.



Execute this example and display the result. Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

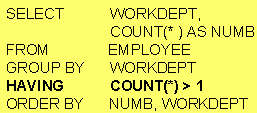
15. GROUP BY, HAVING (2 of 2) -



Execute this example and display the result. Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

**Display the department numbers with more than one employee.**



Use a Snippit to document your SQL output =>

Use a Snippit to document your SQL statement =>

In SQL, what’s the difference between the having clause and the where clause? <http://www.programmerinterview.com/index.php/database-sql/having-vs-where-clause/>

16. What are the differences between the use of a WHERE clause and a HAVING clause? Answer =>

## 5.6 Another GROUP BY and Having Example

Example **SELECT FDEP, COUNT(\*) FROM FACULTY**

**GROUP BY FDEP;**

Report the number of faculty members in each department. The field that will be used to group or categorize the data will be specified in the SELECT list and the GROUP BY list. The SELECT list must contain only column names specified in the GROUP BY list. For example, the following SQL example will cause a syntax error because FPAY is not in the GROUP BY list:

Example  **SELECT FDEP, AVG(FPAY) FROM FACULTY**

**GROUP BY FDEP;**

Report the average faculty pay by department listing departments in insertion order (no order).

Example **SELECT FDEP, AVG(FPAY) FROM FACULTY**

**GROUP BY FDEP**

**ORDER BY FDEP;**

Report the average faculty pay by department listing the department names in alphabetical order.

Example **SELECT FDEP, AVG(FPAY) FROM FACULTY**

**WHERE FPAY > 30000**

**GROUP BY FDEP;**

***Before averaging the faculty pay for each department***, SELECT only those records that have FPAY > 30000. Faculty members earning less than 30000 are not included in the calculation of the departmental average.

Example **SELECT FDEP, AVG(FPAY) FROM FACULTY**

**GROUP BY FDEP**

**HAVING AVG(FPAY) > 40000;**

After averaging all the faculty member's pay for each department, report only those departments that have a departmental average pay greater than 40000. The WHERE clause will limit the selection of records to be averaged. The WHERE clause should only contain field names, logical operators, etc., but no aggregate functions. The HAVING clause can only specify SQL function names, logical operators and constants. Simple column field names are NOT permitted in a HAVING clause.

Example **SELECT FDEP, AVG(FPAY) FROM FACULTY**

**WHERE FPAY > 30000**

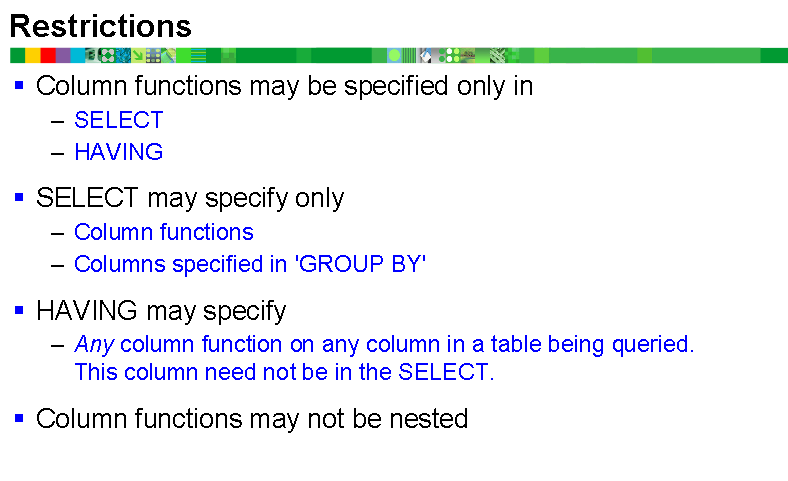
**GROUP BY FDEP**

**HAVING AVG(FPAY) > 40000**

**ORDER BY FDEP;**

Before averaging the faculty pay for each department, SELECT only those records that have FPAY > 30000. After averaging all faculty members' pay for each department, report only those departments that have a departmental average pay greater than 40000. Then list each department's name in alphabetical order.

17. The following table lists SQL statements illustrating popular errors in the use of the GROUP BY and HAVING clauses, and Aggregate (Column) Functions. Using the following table CLEARLY describe each error.



|  |  |
| --- | --- |
| ***I*ncorrect GROUP BY or Aggregate Function Error** | **What is wrong with these SQL statements?** |
| **SELECT EMPNO, WORKDEPT, SUM(SALARY)**  **FROM EMP**  **GROUP BY WORKDEPT**  **HAVING SUM(SALARY) > 30000** |  |
| **SELECT WORKDEPT, AVG(SALARY)**  **FROM EMP**  **WHERE AVG(SALARY) > 25000**  **GROUP BY WORKDEPT** |  |
| **SELECT WORKDEPT, SUM(SALARY)**  **FROM EMP**  **WHERE SEX="F"**  **GROUP BY WORKDEPT**  **HAVING EDLEVEL >18** |  |
| **SELECT WORKDEPT, SUM(SALARY)**  **FROM EMP**  **WHERE SEX="F"**  **HAVING SUM(SALARY) > 25000** |  |
| **SELECT WORKDEPT, SUM(SALARY)**  **FROM EMP**  **WHERE SEX="F"**  **GROUP BY EDLEVEL**  **HAVING SUM(SALARY) > 25000** |  |