





Oracle Apex 2019


SQL Workbook


Oracle Application Express

 Workspace

 LAZARE01



☐ Remember workspace and username 

Sign In

[Reset Password](#)

**Welcome to Leeds Beckett University Oracle 19c running
Apex 19.1.0.00.15**
PLEASE NOTE:
Your *Workspace* is your C number with a **CAPITAL C**
Your *Username* is your C number with a **CAPITAL C**

Student name:

E-mail address

Course

Group

Module tutor

E-mail address

Telephone
number

Table of Contents

Contents

Check out these websites:	4
1. How to use this workbook	5
2. Login into Oracle APEX Application	6
3. Introduction to the Oracle APEX Environment	7
4. SQL Scripts Page	7
4.1 SQL scripts.....	7
5. Creating tables and inserting data using the Object Browser	9
5.1 Dropping a Table	15
5.2 Editing a Table	16
5.3 Summary	16
6. SQL in Apex	17
6.1 Using the SQL Command Editor	18
6.2 Retrieving all the data from a single table	19
6.3 Retrieving data from specific columns from a single table (Projection)	20
6.4 Retrieving data from specific rows from a single table (Selection).....	21
6.5 Selecting data from more than one table.....	22
6.6 COUNT, NULL, BETWEEN, ORDER BY, IN, LIKE.....	23
6.7 SQL Functions	25
6.8 Using Sub queries	26
6.9 Group By / Having	30
6.10 Using Self Joins	32
6.11 Outer Joins	33
7. Views	34
8. SQL Quiz Application	36
Appendix A: Scott Tables	37
Appendix B: SQL tutorial – Solution	39
Appendix C: SQL & PL/SQL Quick Reference Guide	42
Appendix D: Trouble Shooting Common APEX Oracle Error Messages	50
Common Error messages when creating or dropping tables.....	52
1. Invalid table name (ORA-00903)	52
2. Invalid column name (ORA-00904)	52
3. Invalid data type (ORA-00902)	53
4. Table or view does not exist (ORA-00942).....	53
Common Error messages when populating tables	54
1. Missing left parenthesis (ORA-00906).....	54
2. Missing right parenthesis (ORA-00907).....	54
3. Too many values (ORA-00913).....	55
4. Missing comma (ORA-00917).....	55
5. Inserted value too large for column (ORA-01401 and ORA-12899)	56
6. Value larger than specified precision allows for this column (ORA-01438) ..	56
7. Not enough values (ORA-00947)	56
8. Cannot insert NULL into (ORA-01400).....	57
9. Integrity constraint (ORA-02291).....	57
10. Unique constraint (ORA-00001).....	58
11. Quoted string not properly terminated (ORA-01756).....	58
Common Error messages when executing queries	58

1. Missing Expression (ORA-00936).....	58
2. Column ambiguously defined (ORA-00918).....	59
3. Not a GROUP BY expression (ORA-00979)	59
Appendix E: Apex and SQL Documentation.....	61
Introduction to the Oracle APEX environment.....	61
What are SQL Scripts?.....	61
Creating tables and inserting data using Object Browser	62
Primary Key	62
Foreign key	64
Example of creating a FK in SQL script.....	64
ALTER Table FK.....	65
UNIQUE constraint.....	65
CHECK constraint	65
Editing a Table in Object Browser.....	66
Browsing a Table	67
How to create a Composite/Compound Primary Key.....	67
How to create a Composite/Compound Foreign Key	68
Creating a table which has a recursive relationship	68
SQL Commands Enviroment	70
Switching to Another SQL Workshop Component	71
Using the SQL Command Editor	73
Retrieving all the data from a single table	74
Retrieving data from specific columns from a single table (Projection)	74
Retrieving data from specific rows from a single table (Selection).....	75
Join - Selecting data from more than one table.....	76
The LIKE Condition	77
Using Subqueries	79
EXISTS Condition	80
IN Condition	80
Using Self-Joins	80
Group By/Having	81
Outer Joins.....	82
Table Expression.....	84
VIEWS.....	85

Note: These icons are used throughout this workbook:



An example showing how to do something



A task for you to complete



Good to know!



An example and a task for you to complete

Check out these websites:

ApexLib - Feature Demonstration

<http://apex.oracle.com/pls/otn/f?p=33231:41:389827641840101::NO:41::>

ORACLE Application Express Documentation
Release 4.2

http://docs.oracle.com/cd/E37097_01/doc/index.htm

1. How to use this workbook

This workbook should be used during your tutorial and for self study, it relates to topics covered in the lectures. We suggest that you also use the lecture slides to help you with the tutorial tasks.

The weekly breakdown in the module guide will help you to plan your work when working independently.

You are expected to complete all tasks in this workbook.

The following Appendices should be referred to when learning new SQL and PL/SQL topics, you will find valuable theory, examples and solutions on the different topics.

Appendix A has the full SQL code for tables used in this workbook; emp and dept (scott database).

Appendix B contains solutions for most of your SQL tasks.

Appendix C is an SQL & PL/SQL Quick Reference Guide, please check it out, it will be useful for you throughout your db studies, on all levels.

Appendix D is a guide to the most common error messages in Oracle, on the server side. Could be helpful in the beginning.

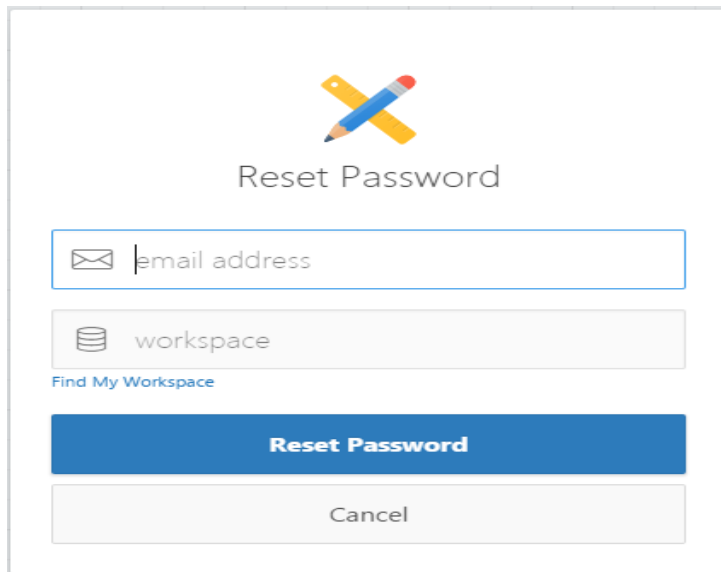
Appendix E explains, why a sub query, or what is a primary key, when to use a GROUP BY command, and so on.

2. Login into Oracle APEX Application

To log in enter your **workspace**, **username** and **password** and then click Login. Workspace and username are your C student ID, e.g. C7777777. These fields are case sensitive, so make sure you use an uppercase 'C'.

If you have forgotten your password you can click on the Reset Password link on the right on the screen.

To reset your password, type in your student email address (you'll find it in your Portal email account) and your username is your C student ID, e.g. C7777777 (upper case 'C'). Then click Reset Password.

The image shows a 'Reset Password' dialog box. At the top is a logo consisting of two crossed pencils, one yellow and one blue. Below the logo is the text 'Reset Password'. There are two input fields: the first is for 'email address' with an envelope icon, and the second is for 'workspace' with a database icon. Below the 'workspace' field is a link that says 'Find My Workspace'. At the bottom are two buttons: a blue 'Reset Password' button and a grey 'Cancel' button.

An email is then sent to your student email account with details of your workspace, username and password. Go back to the log in page and log in using a C student ID for workspace and the username and the provided password in the email. You will be prompted to change your password.

WARNING! if you click more than 3 times on the login button, your account will be locked! To unlock your account, you will have to see someone in a help desk, JG202. Your Module Tutor will not be able to unlock your account.

If you want to use Oracle APEX **19.1.0.00.15** at home or on a personal laptop, visit this website to download Application Express for free: <http://apex.oracle.com>

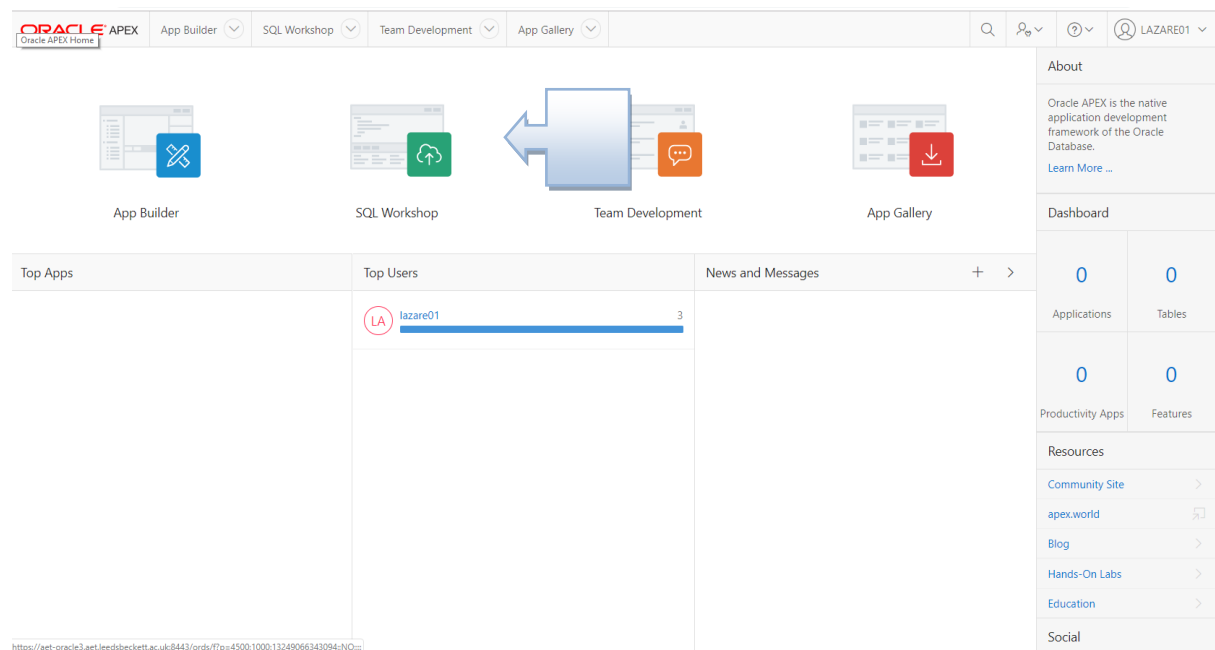
3. Introduction to the Oracle APEX Environment

Oracle Application Express (APEX 5) is a rapid application development tool for producing web applications using an ORACLE database.

The applications can be built using a web browser with only limited programming experience, although you will need to be familiar with ORACLE SQL. APEX allows you to quickly develop professional looking applications which are fast and secure.

4. SQL Scripts Page

On this page you can create a new SQL script and type in all the DDL (e.g. CREATE, DROP) statements and DML (INSERT, UPDATE, DELETE) statements. There is also an option to write your SQL statements (SQL queries), however, it is the SQL Commands page that you will be using for this. The SQL Scripts page displays an interactive report of all SQL scripts created by you as a user.



4.1 SQL scripts

What are SQL Scripts?

An SQL script is a set of SQL commands saved as a file in the SQL Scripts tool. An SQL script can contain one or more SQL statements or PL/SQL blocks. You can use SQL Scripts Environment in Apex to create, edit, view, run, and delete database objects (tables, views and triggers).


To access SQL Scripts:

1. Log in to the Workspace home page.
2. Click the **SQL Workshop** icon and then **SQL Scripts** to drill-down to the SQL Scripts page.





Exercise 1: How to upload a SQL script from your local file system

1. On the Workspace home page, click **SQL Workshop** and then **SQL Scripts**.
2. Click the **Upload** button.
The Upload Script dialogue appears.
3. To upload a script:
 - First you must copy the code from Appendix A into a notepad file, and save it as scott.sql. Use the **Choose File** button, find and select the script you've saved - "scott.sql". Another option is to download the script from Modules page on the VLE and save it to your drive.
 -
4. Give a name to your script. For example: **Scott** -- hopefully you haven't already got a script with this name.
5. Click **Upload** to add the script to the Script Repository.

NOTE: Occasionally a symbol looking similarly like this , is automatically added. All you need to do, is look for it at the beginning of the script or towards the end. When you find it, delete it. Save the changes


6. Run the script to create the tables. The Run Script page appears.
Click **Run Now** to submit the script for execution.


Run Script

 You have requested to run a script containing statement(s) SQL Workshop will ignore. Please confirm your request.

Script Name	scott
Created	on 08/20/2015 09:35:54 AM by LAZARE01
Updated	on 08/20/2015 09:35:54 AM by LAZARE01
Number of Statements	35
Script Size in Bytes	4,015

Line Number	Unknown Statement
69	



7. Click the **View Results** icon for the results you want to view. The **View Results** icon is to the right of the scripts listed on the Manage Scripts Results page. Select **Summary** or **Detail** View, then click **Go**.

- If you receive the message **ORA-000942: table or view does not exist**, go to the Appendix D (*Guide to the most common error messages in Oracle*) to check what it means.
- If you go to the SQL Workshop and choose **Object Browser**, you will be able to see the created tables EMP, DEPT and SALGRADE. If you click on the **table** tab you can view table details such as DATA, CONSTRAINTS, TRIGGERS and much more. Take some time to investigate this environment.

The screenshot shows the Oracle Object Browser interface. On the left, a list of tables is displayed, with 'DEPT' selected. The main area shows the details for the 'DEPT' table, including columns, data types, nullability, and primary keys.

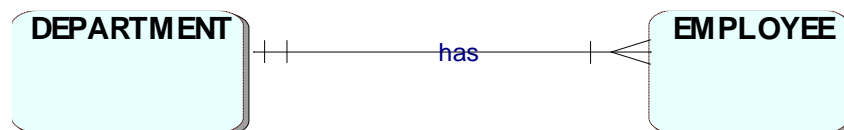
Column Name	Data Type	Nullable	Default	Primary Key
DEPTNO	NUMBER(2,0)	No	-	1
DNAME	VARCHAR2(14)	Yes	-	-
LOC	VARCHAR2(13)	Yes	-	-

5. Creating tables and inserting data using the Object Browser

A table is a unit of data storage in an Oracle database containing rows and columns. Each table must have a table name, column names, data types for each column, a size and any constraints on the table or column that are necessary.

You can create a database objects such as tables in the **Object Browser(OB)**, SQL Scripts or SQL Commands in the Apex Environment. In the next exercise you will create a table using OB, though we advise you to use the SQL Scripts environment and type in your commands, as in the SQL script that you've just run and similar to what you did in your year 1 database module.

For the next two Tasks you will be asked to create two tables in OB, Department and Employee. The ERD shows the relationship between the entities.

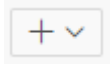


Exercise 2: Create a table called *department*

To create a table using the Object Browser:

- On the Workspace home page, click **SQL Workshop** and then **Object Browser**.

The Object Browser appears.



2. Click Icon plus to **Create**.
3. From the list of object types, select the **Table** object.
4. Enter the table name: **DEPARTMENT**

Table names must conform to Oracle naming conventions and **cannot contain** spaces or start with a number or underscore. To have the final table name match the case entered in the Table Name field, you click on Preserve Case, **do not** select this option for this task.

Column Name	Type	Precision	Scale	Not Null	Identity	Move
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v
	- Select Datatype -					^ v

5. Enter details for each column. For each column:
 - a. Enter the column names **dept_id**, **dept_name**, **loc** and **phone_number**
 - b. Select the column type. For this task select **VARCHAR2** for all columns apart from Dept_id which should be **NUMBER** datatype.
 - c. Enter the following additional information as appropriate:

In the **precision** column (Size of the column) enter dept_id (3), and **scale** columns for dept name(10), loc(10) and phone number (15)

- d. To specify a column that must always have a value, select the check box in the **Not Null** column. Select this check box for all columns apart from dept_id.
- e. **New to Apex 5** is the addition of a column "Identity". Select the identity option (Always the PK) for the column dept_id – which must be of a datatype Number.

*Options include: **Always** - The Oracle Database always uses the sequence generator to assign a value to the column. If you attempt to explicitly assign a value to the column using INSERT or UPDATE, then an error is returned. **Default** - You will have to explicitly assign a specified value to the PK column. **Default on Null** - The Oracle Database uses the sequence generator to assign a value to the column when a subsequent INSERT*

statement attempts to assign a value that evaluates to NULL. This means that you can enter a value for a PK column, or allow the sequence generator to assign a pk column.

Note: To change the order of previously entered columns, click the **Up** and **Down** arrows in the Move column. To add additional columns, click **Add Column**.

- a. Click **Next**.



This table could also be created by typing the following SQL Command directly into the SQL commands Tool window or in the SQL script environment.

DDL (Data Definition Language) code to create a employee table:

```
CREATE TABLE DEPARTMENT (  
dept_id NUMBER(3) PRIMARY KEY,  
dept_name VARCHAR2(10),  
loc VARCHAR2(10),  
phone_number VARCHAR2(15));
```

6. On the Primary Key Page, select **Populate from a new sequence**, and select primary key below dept id leaving the others as defaults.
7. Then click **Next**.

For further information on this topic, please go to [primary key](#) (ctrl + click to follow the link).



WHAT IS A SEQUENCE ?

IMPORTANT:

A sequence generates a serial list of unique numbers for numeric columns of a database table. Database sequences are generally used to populate table primary keys. When a sequence number is generated, the sequence is incremented, when a user enters a new row of data. For example: If your student id is 1111, and sequence increment is by 1 (which is by default), the next id that will be assigned by a system will be 1112.

To create a sequence in SQL Scripts, you could use DDL code to CREATE and DROP a sequence. See scott.sql script code for its code, at the end of the

script. Normally specified at the end of a script, if script contains the INSERT commands as well, otherwise it's specified after CREATE TABLE commands.

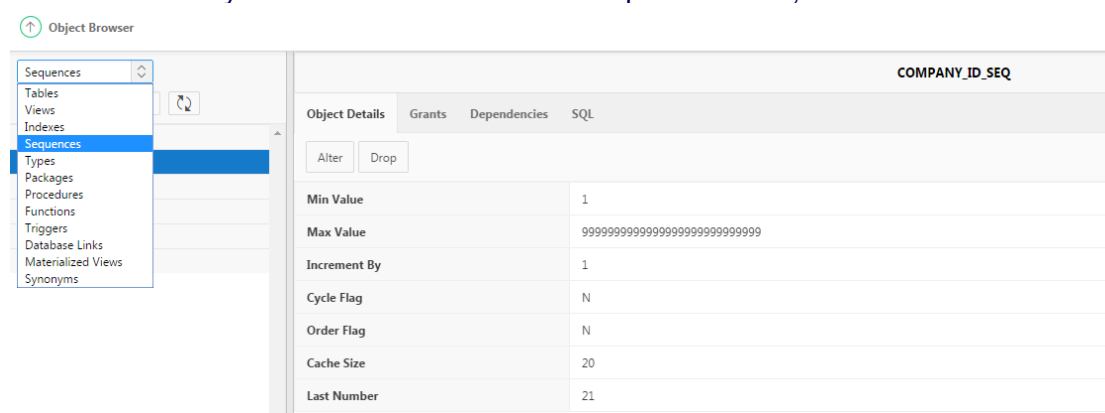


```
Drop sequence seq_name;  
CREATE SEQUENCE seq_name;
```

Example for department table:

```
DROP SEQUENCE dept_seq;  
CREATE SEQUENCE dept_seq start with 50;
```

Note that you can create and browse sequence in OB, as shown below:

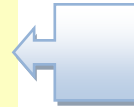


8. On the Foreign Key(FK) page, **click Next**. The DEPARTMENT table hasn't got a FK, therefore there is no need to declare one, however there will be one on the EMPLOYEE table, see later section.
9. **Add a UNIQUE** constraint for the **dept_name** column.
 - a. Specify the **Constraint Type**, select **Unique**
 - b. Select attribute/column name **DNAME** and move it into "**Key Column**"
 - b. Make sure ***Name (Constraint name)** is unique: Enter DName_Unique (if you get an error message: "**Constraint name already used**", then you will have to define a different name, for example add a number to DName_Unique2.
 - c. Click **Add**.
10. Click **Next**. A confirmation page appears.
11. Click **Create Table!**



 You can define a UNIQUE constraint within the SQL scripts code and it would look like this example:

```
CREATE TABLE DEPARTMENT (
  Dept_id NUMBER (3) PRIMARY KEY,
  dept_name VARCHAR2 (10) UNIQUE,
  loc VARCHAR2 (10),
  phone_number VARCHAR2 (15));
```



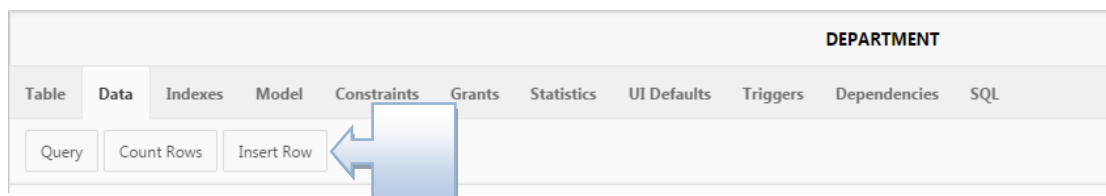
For further information on this topic, please go to **unique constraint** (ctrl + click to follow the link).



Exercise 3: Insert data into the DEPARTMENT table

To insert data into the DEPARTMENT table:

1. Click on the tab 'Data' and use 'Insert Row' to insert new data into the table.



You can use this data:

Dept ID	Dept Name	Loc	Phone
Leave blank	ACCOUNTING	NEW YORK	11111
Leave blank	RESEARCH	DALLAS	22222



Exercise 4: Create a table called EMPLOYEE

1. In the **Object Browser (OB)**, following the same steps as when creating department table, create a table called **EMPLOYEE**, with column definitions described below:

```
CREATE TABLE EMPLOYEE (
  EMPNO NUMBER (4) PRIMARY KEY,
  ENAME VARCHAR2 (10),
  JOB VARCHAR2 (9),
  MGR NUMBER (4),
  HIREDATE DATE,
  SAL NUMBER (7,2),
  COMM NUMBER (7,2),
  DEPTNO NUMBER (3) REFERENCES department (Dept_id));
```

Note 1: that in Object Browser size (7,2) for the COMM and SAL column is specified as number 2 in the scale column, and 7 in a precision column.

Note 2: This table has a FK column. Remember that FK column(s) must have same datatype and size as the PK column in the referenced table, but not the FK column name, as in the above example.

2. To Add a Foreign Key (FK) you can specify it while creating a table in OB, or using the constraint tab after you have created a table. The example below is creates a FK while you are creating the table:
 - a. Enter a **name** of the foreign key constraint that you are defining, in this case enter: fk1_deptno
 - b. Select **Cascade Delete** - Deletes the dependent rows from this table when the corresponding parent table row is deleted.
 - c. Select **Key Column** - Select the column **deptno**, then click the **Add** icon to move them to Key Column(s).
 - d. **References Table** - Select the **department** table which is referenced by this foreign key. Then, select **dept_id** referenced by this foreign key. Once selected, click the **Add** icon to move the selected columns to Referenced Column(s).
 - e. Now, Click Next.

On how to Create a FK in OB under Constraints tab, go to **foreign key** section in the Appendices. In this section you can also learn on how to define a FK within a table or how to use ALTER table command.

3. What other constraints could you implement - perhaps a **CHECK** constraint for the Salary and hire_date columns?



CHECK constraints code examples written within an SQL script or in OB.

For column **Status**:

This command would be written within an SQL script:
STATUS IN ('OPEN','CLOSED','PENDING')

This is how it would be specified in the OB:

Constraint Type ☒ Check ☐ Unique

Check Condition

'OPEN','CLOSED','PENDING'

For columns Salary and Commission, to make sure a value is entered in either salary or commission:

`SAL IS NOT NULL OR COMM IS NOT NULL`

For column Employee name, for example name cannot be longer than 6 characters:

`LENGTH(ENAME) <= 6`

For column Salary, it has to have value greater than 1:

`SAL > 0`

Research what do the following commands do:

`SAL + NVL(COMM,0) > 1000`

`= UPPER(ENAME)`

`(SAL * 100) = trunc(SAL * 100)`

`HIREDATE > SYSDATE - 7`

4. Insert data into the table. For example:

`INSERT INTO EMPLOYEE VALUES (' ', 'SMITH', 'CLERK ', 7902,
'12/08/13', 800, null, 1);`

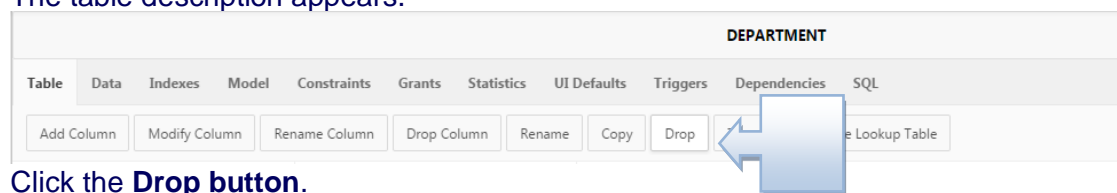
`INSERT INTO EMPLOYEE VALUES (' ', 'KING', 'PRESIDENT', null,
'08/08/13', 5000, null, 1);`

5.1 Dropping a Table

To drop a table:

1. On the Workspace home page, click **SQL Workshop** and then **Object Browser**.
Object Browser appears.
2. From the Object list, ensure **Table** is selected.
3. From the Object Selection pane, select a table. For this task, select one of the tables you have created before this session (in year 1 database module).

The table description appears.



4. Click the **Drop** button.
5. Tick the box with **Cascade Constraints**.
6. To confirm, click **Finish**.



If you have created your tables using SQL scripts, your drop table command would be specified at the top of the SQL script, like this:

```
DROP TABLE department CASCADE CONSTRAINTS;  
DROP TABLE employee CASCADE CONSTRAINTS;
```



Why and when do we have to specify CASCADE CONSTRAINTS after the table name?

5.2 Editing a Table

While viewing a table description, you can edit it by clicking the buttons above the table description.

To edit a table:

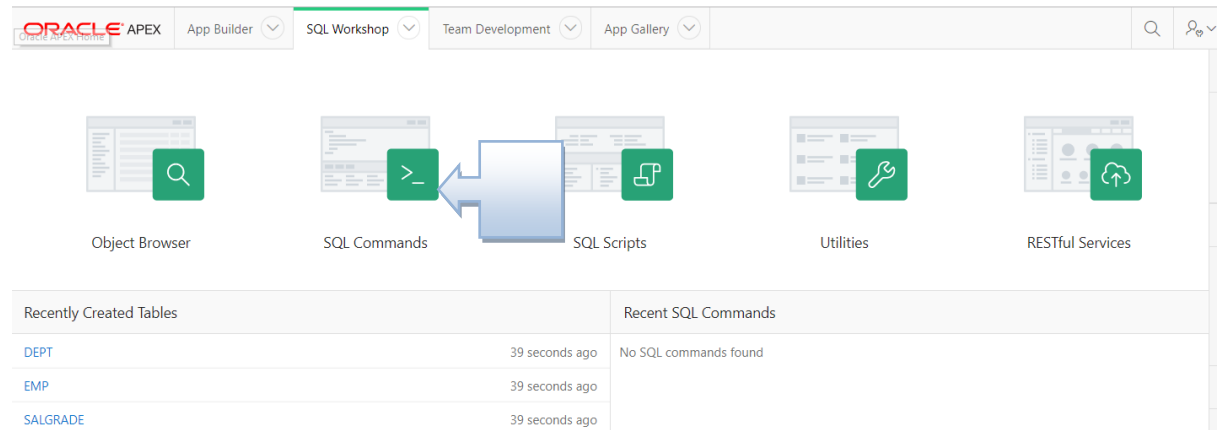
1. On the Workspace home page, click **SQL Workshop** and then **Object Browser**.
Object Browser appears.
2. From the Object Selection pane, select a table, for this task, select **EMPLOYEE**. The table description appears.
3. Click the Modify Column button and change **ename** column name datatype to CHAR and size to 15. Then click Next, then Finish.

5.3 Summary

In this section you have learned how to create tables, define primary and foreign keys, what different column data types there are, how to define CHECK, UNIQUE constraints, assigning NOT NULL, insert dummy data, and code that can be used to create a table in the **SQL Scripts** environment.

Important note If you create your tables in Object Browser and if at some point you decide to drop a table, this table can't be recreated easily (you would have to re-enter the information). However if you create your table(s) using an SQL Script, you will be able to re-create your table and data at any point, by re-running your script. Think about this implication, and how you can save your own time.

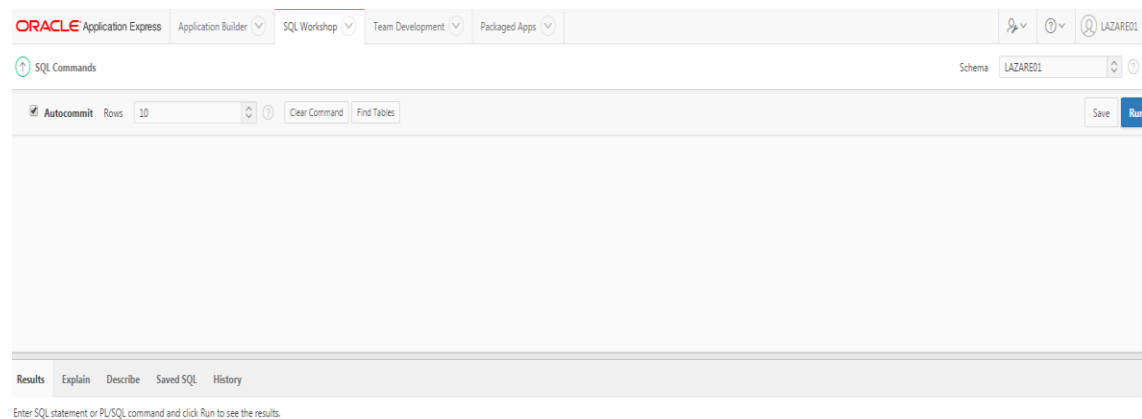
6. SQL in Apex



SQL is the standard language used to create and query Relational Database tables. In the last section you looked at how to create tables, drop them, insert some data and modify data and columns. This section of the document will focus on the SQL statements required to do basic, intermediate and advanced queries on the **scott** database tables: salgrade, emp and dept. Section 9 will focus on PL/SQL and Triggers based on same tables.

In the previous sections you created employee and department. As these haven't got many records, we will use **scott** db.

We suggest that you download a Scott script from the VLE module, found in the SQL materials section. Upload and Run the script in the SQL Scripts environment, unless you've already got your tables set up, as suggested in Exercise 1.



If you would like to learn more about the SQL Commands Home Page, please go to : **About the SQL Commands Home Page**

6.1 Using the SQL Command Editor



Exercise 5: Running an SQL Command

1. To execute an SQL Command:
 - a) On the Workspace home page, click **SQL Workshop** and then **SQL Commands**.
 - b) The SQL Commands page appears.
 - c) Enter the SQL command you want to run in the command editor.

For example type in:

```
SELECT *  
FROM cat;
```

- d) Then click **Run** (or Ctrl + Enter) to execute the command.

Important Note: If you have entered more than one SQL Statement in the SQL Commands window, highlight the statement you want to **Run**.

This is the error message you will get if you try to run two or more SQL commands at the same time.

```
SELECT * FROM emp;
```

```
SELECT * FROM dept;
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------



ORA-00933: SQL command not properly ended

- e) The result appears in the **Results** pane below.
 - f) Click **Save** to save this command. Call it **DD_query**.
 - g) Find out what does **CAT** stand for in ORACLE APEX. Is this a table?
2. To export the resulting report as a comma-delimited file (.csv) file, click the **Download** link. You do not need to do this task now.
3. **Saved SQL.** From a saved SQL command tool, click the **Saved SQL** tab to display a list of all saved SQL commands, in the current workspace. For example click on the **DD_query** title to load it into the command editor.

6.2 Retrieving all the data from a single table

Retrieving all the data (all columns and rows of data) from a single table is the simplest data selection.

The following syntax defines the **SELECT** statement:

SELECT *
FROM tablename;



Exercise 6

To execute an SQL Command:

1. On the Workspace home page, click **SQL Workshop** and then **SQL Commands**.
2. The SQL Commands page appears.
3. Enter the SQL command, in the box as below, to run in the command editor.

```
SELECT *  
FROM dept;
```

4. Click **Run** (or Ctrl+Enter) to execute the command.

Please Note:

You must leave spaces in between each of the words in your SQL statements, if you don't SQL will not be able to understand the statement and will produce an error. **Check Appendix D for the errors guide.**



Exercise 7

To select all the data from the **dept** table you could also enter:

```
SELECT deptno, dname, loc  
FROM dept;
```

Which would produce exactly the same results as **SELECT * FROM dept;**

The ***** operator provides a shorthand way of specifying all of the columns of data.



Exercise 8

Write an SQL command to return all the data from the emp table to achieve the following output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/0080	800	-	20
7499	ALLEN	SALESMAN	7698	02/20/0081	1600	300	30
7521	WARD	SALESMAN	7698	02/22/0081	1250	500	30
7566	JONES	MANAGER	7839	04/02/0081	2975	-	20
7654	MARTIN	SALESMAN	7698	09/28/0081	1250	1400	30
7698	BLAKE	MANAGER	7839	05/01/0081	2850	-	30
7782	CLARK	MANAGER	7839	06/09/0081	2450	-	10
7788	SCOTT	ANALYST	7566	04/19/0087	3000	-	20
7839	KING	PRESIDENT	-	11/17/0081	5000	-	10
7844	TURNER	SALESMAN	7698	09/08/0081	1500	0	30
7876	ADAMS	CLERK	7788	05/23/0087	1100	-	20
7900	JAMES	CLERK	7698	12/03/0081	950	-	30
7902	FORD	ANALYST	7566	12/03/0081	3000	-	20
7934	MILLER	CLERK	7782	01/03/0082	1300	-	10

Save the script as **select2**.

6.3 Retrieving data from specific columns from a single table (Projection)

When selecting data from a table we may not want to select all of the data from the table as this may result in a cluttered output which could be confusing to read. Only specify the columns that you are interested in.



To return a specific column from a table you could write a statement like this:

```
SELECT dname  
FROM dept;
```



Exercise 9

Write an SQL command to display all different job types.

Tip: use emp table

Run the command and check the results. What different job types are there? Save your query as projection1.

6.4 Retrieving data from specific rows from a single table (Selection)

When selecting data it is unusual to retrieve every row (or every column of every row) of a large table. More often, rows are retrieved according to a value or range of values held in one or more columns of the table.



Exercise 10

In the SQL Command environment, write and Run the SQL command below (selection1). Check the results.

```
SELECT *  
FROM emp  
WHERE empno = 7369;
```



Exercise 11

Modify your query 9 (projection1) to select **names** and **job** for employees in the **department number 20**. This will be a SELECT statement. Save your query as selection2.

The output should look like this:

Results Explain Describe Saved SQL History	
ENAME	JOB
SMITH	CLERK
JONES	MANAGER
SCOTT	ANALYST
ADAMS	CLERK
FORD	ANALYST

5 rows returned in 0.02 seconds [Download](#)



Exercise 12: Challenge

How could you modify the query above to display “Employee Job Title is”? You will need to use a **Concatenation Operator**. Check your lecture slides. Save your query as concatenation1.

6.5 Selecting data from more than one table

To select the data from more than one table, you list the columns required in the **SELECT** statement, list the tables required in the **FROM** clause, specify the join between the tables (via the Primary and Foreign Keys) in the **WHERE** clause and set the selection criteria using an additional **AND** statement.

The following example shows a join in a query.

```
SELECT ename, job, dname
FROM emp, dept
WHERE dept.deptno=emp.deptno;
```

This equijoin returns details of what has been selected by the SELECT statement. The output is:

Results

Explain

Describe

Saved SQL

History

ENAME	JOB	DNAME
SMITH	CLERK	RESEARCH
ALLEN	SALESMAN	SALES
WARD	SALESMAN	SALES
JONES	MANAGER	RESEARCH
MARTIN	SALESMAN	SALES
BLAKE	MANAGER	SALES
CLARK	MANAGER	ACCOUNTING
SCOTT	ANALYST	RESEARCH
KING	PRESIDENT	ACCOUNTING
TURNER	SALESMAN	SALES

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.01 seconds

Download



Exercise 13

Write an SQL command to modify the query below to return only records for all staff managed by employee name KING. Run and Save the script as simplejoin1.

```
SELECT ename, job, dname  
FROM emp, dept  
WHERE Dept.deptno=emp.deptno;
```



Exercise 14

Show all employees working at location - **Dallas**. Run and Save your query as simplejoin2.sql



Exercise 15

Modify your query 12 (concatenation1) and select also **empno** only for those that are managed by mgr **7839**. Run and Save this query as query15.

6.6 COUNT, NULL, BETWEEN, ORDER BY, IN, LIKE



Exercise 16

Using the COUNT function, find out how many employees are there in department number 20. Run and Save this query as count1.



Exercise 17

Using a DISTINCT function list different job title. Save this query as distinct1. Run and Save this query as count1. Note that DISTINCT is used to eliminate duplicate values in a column.



Exercise 18

Select all employees that have no commission. You will need to use IS NULL function. Run and Save this query as Isnull1.



Exercise 19

Modify your query 12 to include department names. Delete condition for a specific manager in the statement. Run and Save this query as join1. The report would look like this:

ENAME 'EMPLOYEEJOB TITLE IS' JOB	DNAME
SMITH Employee Job Title is CLERK	RESEARCH
ALLEN Employee Job Title is SALESMAN	SALES
WARD Employee Job Title is SALESMAN	SALES
JONES Employee Job Title is MANAGER	RESEARCH
MARTIN Employee Job Title is SALESMAN	SALES
BLAKE Employee Job Title is MANAGER	SALES
CLARK Employee Job Title is MANAGER	ACCOUNTING
SCOTT Employee Job Title is ANALYST	RESEARCH
KING Employee Job Title is PRESIDENT	ACCOUNTING
TURNER Employee Job Title is SALESMAN	SALES
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.02 seconds [Download](#)

The title for this output is not very good. How could you change it so it looks like this?

Employee Job Title	DNAME
--------------------	-------



Exercise 20

Modify previous query, again, to include staff only in the **SALES** department.



Exercise 21

List all employees who have a salary **between 1000 and 2000**. Use the BETWEEN operator. Save query as between1.



Exercise 22

List the details of the employees in **departments 10 and 20** in alphabetical order by name. Save query as Order1.



Exercise 23

Display all employee names which have TH or LL in them. Use LIKE operator here. Save query as like1.



Exercise 24

Display name, annual salary and commission of all salesmen whose monthly salary is greater than their commission. The output should be ordered by salary, highest first, and then if two or more employees have the same salary, by employee name.

Results	Explain	Describe	Saved SQL	History
ENAME	ANNUAL_SAL	COMM		
ALLEN	19200	300		
TURNER	18000	0		
WARD	15000	500		

3 rows returned in 0.02 seconds [Download](#)

6.7 SQL Functions



Exercise 25

List employee name, job, salary, grade and department name for everyone in the company except the clerks, sort on salary highest first.

This is the pseudo code:

```
SELECT ename, job, sal, grade, dname
FROM table1, table2, table 3
WHERE      table.PK= table.FK
AND  col BETWEEN losal and hisal
AND  col != 'value'
ORDER BY sal DESC;
```

Your output should look like this. Save query as join3tables.sql.

Results	Explain	Describe	Saved SQL	History
ENAME	JOB	SAL	GRADE	DNAME
KING	PRESIDENT	5000	5	ACCOUNTING
FORD	ANALYST	3000	4	RESEARCH
SCOTT	ANALYST	3000	4	RESEARCH
JONES	MANAGER	2975	4	RESEARCH
BLAKE	MANAGER	2850	4	SALES
CLARK	MANAGER	2450	4	ACCOUNTING
ALLEN	SALESMAN	1600	3	SALES
TURNER	SALESMAN	1500	3	SALES
MARTIN	SALESMAN	1250	2	SALES
WARD	SALESMAN	1250	2	SALES
More than 10 rows available. Increase rows selector to view more rows.				
10 rows returned in 0.03 seconds Download				



Exercise 26

Find the difference between the highest and lowest salaries.

Use the following syntax for this task:

```
SELECT max(col) - min(col) OutputTitle  
FROM table;
```

Save your query as MinMax.sql

6.8 Using Sub queries

A **sub query** answers, multiple-part questions. For example, to determine who works in Taylor's department, you can first use a sub query to determine the department in which Taylor works. You can then answer the original question with the parent SELECT statement. A sub query in the FROM clause of a SELECT statement is also called an **inline view**. A sub query in the WHERE clause of a SELECT statement is often called a **nested sub query**.



List all employees who have a salary higher than average salary. First of all you need to know what the AVG salary is. The statement **SELECT AVG(sal) FROM emp** will calculate this for you and produce a result of 2073.2142857

This result can be rounded using the **ROUND** function,

```
SELECT ROUND (AVG(sal)) FROM emp
```

This will produce a result of 2073. You can then use this result to produce a list of all employees who have a salary higher than the AVG salary.



```
SELECT deptno, ename, sal FROM emp WHERE sal > 2073
```

DEPTNO	ENAME	SAL
20	JONES	2975
30	BLAKE	2850
10	CLARK	2450
20	SCOTT	3000
10	KING	5000
20	FORD	3000

To sort the output use **ORDER BY**.

In the previous case you had to write two SQL statements to produce the output. What you could have done originally is produce a subquery.

See the example below.

☒ Autocommit Rows 10   Save Run

```
SELECT deptno, ename, sal
FROM emp x
WHERE sal > (SELECT AVG(sal)
             FROM emp
             WHERE x.deptno = deptno)
ORDER BY deptno;
```

Results Explain Describe Saved SQL History

DEPTNO	ENAME	SAL
10	KING	5000
20	JONES	2975
20	FORD	3000
20	SCOTT	3000
30	ALLEN	1600
30	BLAKE	2850

6 rows returned in 0.01 seconds [Download](#)



Exercise 27

List all the departments where there are no employees (using a sub-query).

Operator to be used between sub and main query would be NOT IN.
Run and save your query as subquery1.sql
To help you with the task, the syntax for the query is given below:

```
SELECT col1, col2
FROM table1
WHERE col1 NOT IN
      (SELECT col1
       FROM table2);
```



Exercise 28

Find the most recently hired employees in each department.
You will have to write a sub-query here. GROUP BY will be explained later.

Run and Save your query as subquery2.sql

Syntax

```
SELECT columns...
FROM table
WHERE (col1, col2) IN
      (SELECT max(col1), col2
       FROM table
       Group by col);
```

The following should be output:

DEPTNO	ENAME	HIREDATE
20	ADAMS	05/23/1987
30	JAMES	12/03/1981
10	MILLER	01/23/1982



Exercise 29

Show the following details for all employees who earns a salary greater than the average for their department:

Results	Explain	Describe	Saved SQL	History
ENAME	SALARY	DEPTNO		
ALLEN	1600	30		
JONES	2975	20		
BLAKE	2850	30		
SCOTT	3000	20		
KING	5000	10		
FORD	3000	20		

6 rows returned in 0.01 seconds [Download](#)

Save your query as subquery3.sql



Exercise 30

Find the employee(s) who earn the highest salary in each job type. Run and save your query as subquery4.sql

JOB	ENAME	SAL
SALESMAN	ALLEN	1600
MANAGER	JONES	2975
ANALYST	SCOTT	3000
PRESIDENT	KING	5000
ANALYST	FORD	3000
CLERK	MILLER	1300



Exercise 31

Find the employees who earn the minimum salary for their job. Display the result in ascending order of salary.



Exercise 32

Your task is to write a statement to find out who works for the Sales Department, but is not a SALESMAN?

6.9 Group By / Having

Aggregate functions (MAX, AVG, MIN) return a single result row based on groups of rows, rather than on single rows. They are commonly used with the GROUP BY clause in a SELECT statement, where Oracle Database divides the rows of a queried table or view into groups.

You use aggregate functions in the HAVING clause to eliminate groups from the output based on the results of the aggregate functions, rather than on the values of the individual rows of the queried table or view.



Exercise 33

List the minimum and maximum salary for each job type where the output should look like this:

Results	Explain	Describe	Saved SQL	History
JOB	MAXIMUM	MINIMUM		
MANAGER	2975	2450		
ANALYST	3000	3000		
CLERK	800	800		
PRESIDENT	5000	5000		
SALESMAN	1600	1250		
CLERK	1300	950		

6 rows returned in 0.02 seconds [Download](#)

Save your query as groupby.sql

Syntax for retrieving data from a table

```
SELECT column, group function (column)
FROM tablename
GROUP BY column/expression;
```



Exercise 34

Find all departments which have more than 3 employees. To help you with this task, you are given a solution to the newest/hardest part. Run and Save your query as groupbyhaving.sql

```
SELECT col, count(*)
FROM table
GROUP BY col
having count(*) > 3;
```



Exercise 35

List lowest paid employees working for each manager. Exclude any groups where the salary is less than 1000. Sort the output by salary. Use Group By and Having here.

Your output should look like this:

Results	Explain	Describe	Saved SQL	History
MGR	MIN(SAL)			
7788	1100			
7782	1300			
7839	2450			
7566	3000			
-	5000			

5 rows returned in 0.01 seconds [Download](#)

Save your query as groupbyhaving2.sql

6.10 Using Self Joins



Example: The following query uses a self-join to return the name of each employee along with the name of the employee's manager.

☒ Autocommit Rows: 10 Save Run

```
SELECT e1.emp_name || ' works for ' || e2.emp_name "Employees and Their Managers"
FROM employee e1, employee e2
WHERE e1.manager_no = e2.emp_no
ORDER BY e1.emp_name;
```

Results Explain Describe Saved SQL History

Employees and Their Managers
JONES works for SMITH
SANELA works for SMITH
WARD works for SMITH

3 rows returned in 0.01 seconds [Download](#)

The join condition for this query uses the aliases e1 and e2 for the sample table employees: **e1.manager_no = e2.emp_no**



Exercise 36

List all employees by name and number along with their manager's name and number. This is a self-join statement. This statement is most often used on a table with a recursive relationship, as in this case study – Emp table. Run and Save it as selfjoin.sql

Your output should look like this.

EMPNO	ENAME	MGRNO	MGR_NAME
7369	SMITH	7902	FORD
7499	ALLEN	7698	BLAKE
7521	WARD	7698	BLAKE
7566	JONES	7839	KING
7654	MARTIN	7698	BLAKE
7698	BLAKE	7839	KING
7782	CLARK	7839	KING
7788	SCOTT	7566	JONES
7844	TURNER	7698	BLAKE
7876	ADAMS	7788	SCOTT
7900	JAMES	7698	BLAKE
7902	FORD	7566	JONES
7934	MILLER	7782	CLARK

13 rows returned in 0.00 seconds [Download](#)

6.11 Outer Joins



Exercise 37

Write a statement to display all employees by name and number along with their manager's name and number, including KING who hasn't got a manager. You will need to use the (+) outer join operator for this statement. Save the query as outerjoin1.

You have been given part of the query:

```
SELECT  emps.empno, emps.ename, mgrs.empno mgrno, mgrs.ename,
mgr_name
FROM      emp emps, emp mgrs
WHERE     emps.mgr = mgrs.empno;
```



Exercise 38

Display the department that has no employees. Run and Save the query as outerjoin2.sql



Exercise 39

Display those who manage employees in either department 10 or 20. Order by manager number. Use a UNION join here. Run and Save the query as union1.sql

The syntax is:

```
SELECT col
FROM table
WHERE     col = 'value'
UNION
SELECT col
FROM table
WHERE     col = 'value'
```



Exercise 40

Find any jobs that were filled in the first half of 1983 and were also filled during the same period 1984. Save your query as union2.sql



Exercise 41

Find all employees who joined the company before their manager. Save your query.

7. Views

The **CREATE VIEW** statement creates a view of the tables specified in the Select Query clause. **A view is a logical table that is based on one or more detail tables.** The view itself contains no data. It is sometimes called a non-materialized view to distinguish it from a materialized view, which does contain data that has already been calculated from detail tables.

Syntax:

DROP VIEW *ViewName*;
CREATE VIEW *ViewName* AS *SelectQuery*



```
CREATE OR REPLACE VIEW v_emp
AS
SELECT EMPNO, ename,sal,comm, e.deptno, dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

To view data in the view, you can write an SQL statement:

SELECT * FROM v_emp;

EMPNO	ENAME	SAL	COMM	DEPTNO	DNAME
-----	-----	-----	-----	-----	-----
7369	SMITH			20	RESEARCH
7499	ALLEN	1600	300	30	SALES
7521	WARD	1250	500	30	SALES
7566	JONES	2975		20	RESEARCH
7654	MARTIN	1250	1400	30	SALES
7698	BLAKE	2850		30	SALES
7782	CLARK	2450		10	ACCOUNTING
7788	SCOTT	3000		20	RESEARCH
7839	KING	5000		10	ACCOUNTING
7844	TURNER		1500	0	30 SALES
7876	ADAMS		1100		20 RESEARCH
7900	JAMES		950		30 SALES
7902	FORD		3000		20 RESEARCH
7934	MILLER		1300		10 ACCOUNTING



Exercise 42

View: Create a view to hold non-sensitive data on all employees in Dallas. Use the lecture slides from week 3 to see how to create a view. Once you have created your view, select data for the employees who are managed by manager 7566.

```
DROP VIEW ViewName;
```

```
CREATE VIEW ViewName  
AS Select col2, col2, col3  
From table1, table 2  
Where join ... etc
```

8. SQL Quiz Application

The **SQL Quiz application** was developed to help you practice SQL statements. It is based on Scott tables (see above) and it has around 40 different SQL statements of difficulty level.

The screenshot shows the SQL Quiz application interface. At the top, it says "SQL Quiz: All Questions are based on the scott Database". There are buttons for "View Database Diagram", "Previous", and "Next". Below these, there are input fields for "Question Number" (set to 1), "Level" (set to Basic), and "Operator" (set to All). The "Challenge" text says "Select all the information from the salgrade table".

On the left, a blue arrow points to the "Enter your SQL:" text box with the annotation "1. Type in your query". The text box contains "Select * from emp". Below it is a "Run >" button.

In the center, a blue arrow points to the "Run >" button with the annotation "2. Run it!".

Below the text box, there is a "hide <== Suggested Solution" link. Underneath, the "Suggested Solution:" text box contains "SELECT * FROM salgrade;". A blue arrow points to this box with the annotation "3. Check solution". Below it is another "Run >" button.

On the right, a table of employee data is visible:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7501	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7521	MARTIN	SALESMAN	7839	02-APR-81	2975	-	20
7566	SCOTT	ANALYST	7566	19-APR-87	3000	-	20

How to install SQL QUIZ in your Apex Account:

Complete the following:

- 1) Download from MyBeckett SQL QUIZ.zip to your student drive and unzip the SQL QUIZ.zip
- 2) Log into your APEX account
- 3) Go to SQL WORKSHOP > SQL SCRIPTS > upload > quiz_table_build.sql
- 4) Run Script: quiz_table_build.sql
- 5) Check in OBJECT BROWSER that tables QUIZ_QUESTIONS, EMP, DEPT and SALGRADE have been created
- 6) Go to HOME > APPLICATION BUILDER> IMPORT > BROWSER choose file f110(2).sql then click next > choose "Parsing Schema" your own workspace then click INSTALL
- 7) Run Application SQL QUIZ
- 8) To view ERD go to Application Page > Shared Components > IMAGES > CREATE > UPLOAD (browse..) > ERD.jpg
- 9) ... and there you go

Test the SQL quiz application. Try to answer 2-3 SQL statements of your own choice using the application tool. The tutor will demo it in class.

Appendix A: Scott Tables

```
DROP TABLE DEPT CASCADE CONSTRAINTS ;
DROP TABLE EMP CASCADE CONSTRAINTS ;
DROP TABLE SALGRADE CASCADE CONSTRAINTS ;
```

--commands to create tables

```
CREATE TABLE DEPT (
  DEPTNO NUMBER (2) ,
  DNAME  VARCHAR2 (14),
  LOC   VARCHAR2 (13),
  CONSTRAINT PK_DEPT2 PRIMARY KEY ( DEPTNO ));
```

```
CREATE TABLE EMP (
  EMPNO  NUMBER (4) ,
  ENAME  VARCHAR2 (10),
  JOB   VARCHAR2 (9),
  MGR   NUMBER (4),
  HIREDATE DATE,
  SAL   NUMBER (7,2),
  COMM  NUMBER (7,2),
  DEPTNO NUMBER (2) REFERENCES dept (DEPTNO),
  CONSTRAINT PK_EMP2 PRIMARY KEY ( EMPNO ));
```

```
CREATE TABLE SALGRADE (
  GRADE NUMBER,
  LOSAL NUMBER,
  HISAL NUMBER);
```

-- insert data into dept, emp and salgrade, note salgrade is not linked to emp and dept, however dept is a FK in emp

```
INSERT INTO  DEPT (DEPTNO, DNAME, LOC) VALUES (10,
'ACCOUNTING', 'NEW YORK');
INSERT INTO  DEPT (DEPTNO, DNAME, LOC) VALUES (20, 'RESEARCH',
'DALLAS');
INSERT INTO  DEPT (DEPTNO, DNAME, LOC) VALUES (30,'SALES',
'CHICAGO');
INSERT INTO  DEPT (DEPTNO, DNAME, LOC) VALUES (40,'OPERATIONS',
'BOSTON');
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7369, 'SMITH', 'CLERK ', 7902, '12/17/80', 800, null, 20);
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '02/20/81',1600, 300,
30);
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7521, 'WARD', 'SALESMAN', 7698, '02/22/81', 1250, 500,
30);
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7566, 'JONES', 'MANAGER', 7839, '04/02/81', 2975, null,
20);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7654, 'MARTIN','SALESMAN', 7698, '09/28/81', 1250, 1400,
30);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7698, 'BLAKE', 'MANAGER', 7839, '05/01/81', 2850, null,
30);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7782, 'CLARK', 'MANAGER', 7839, '06/09/81', 2450, null,
10);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES (7788, 'SCOTT', 'ANALYST', 7566, '04/19/87', 3000, null,
20);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7839, 'KING', 'PRESIDENT', null, '11/17/81', 5000, null, 10);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7844, 'TURNER','SALESMAN', 7698, '09/08/81', 1500, 0,
30);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7876, 'ADAMS', 'CLERK', 7788, '05/23/87', 1100, null,
20);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7900, 'JAMES', 'CLERK', 7698, '12/03/81', 950, null, 30);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7902, 'FORD', 'ANALYST', 7566, '12/03/81', 3000, null,
20);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO) VALUES ( 7934, 'MILLER','CLERK', 7782, '01/03/82', 1300, null,
10);
```

```
INSERT INTO salgrade(GRADE, LOSAL,HISAL) VALUES (1, 700,
1200);
```

```
INSERT INTO salgrade(GRADE, LOSAL,HISAL) VALUES (2, 1201,
1400);
```

```
INSERT INTO salgrade(GRADE, LOSAL,HISAL) VALUES (3, 1401,
2000);
```

```
INSERT INTO salgrade(GRADE, LOSAL,HISAL) VALUES (4, 2001,
3000);
```

```
INSERT INTO salgrade(GRADE, LOSAL,HISAL) VALUES (5, 3001,
9999);
```

```
-- this command will add FK constraint to emp table, based on the recursive
relationship
```

```
ALTER TABLE emp ADD CONSTRAINT fk_manager FOREIGN KEY
(mgr)REFERENCES emp(empno);
```

```
-- this command is to create and drop sequences for PK columns in emp and dept
tables
```

```
DROP sequence dept_seq;
```

```
CREATE sequence dept_seq start with 50;
```

```
DROP sequence emp_seq;
```

```
CREATE sequence emp_seq start with 8000;
```

```
COMMIT;
```

```
/
```

Appendix B: SQL tutorial – Solution

8. select * from emp
9. select job from emp
10. none
11. select ename, job from emp where deptno = 20
12. select ename || ' Employee Job Title is ' || job from emp where deptno = 20
13. select e.ename from emp e, emp m where m.empno = e.mgr and m.ename = 'KING'
14. select ename, sal, loc location from emp, dept where emp.deptno = dept.deptno and loc = 'DALLAS';
15. select empno, ename || ' Employee Job Title is ' || job from emp where deptno = 20 and mgr = 7839.
16. select COUNT(*) from emp where deptno = 20
17. select DISTINCT(job) from emp
18. select ename, job from emp where comm IS NULL
19. select ename || ' Employee Job Title is ' || job **AS "Employee Job Title "**, dname from emp, dept where emp.deptno=dept.deptno
20. select ename || ' Employee Job Title is ' || job **AS "Employee Job Title "**, dname from emp, dept where emp.deptno=dept.deptno and dname = 'SALES'
21. select ename, deptno, sal from emp where sal between 1000 and 2000
22. select * from emp where deptno in (10,20) order by ename;
23. select ename from emp where ename like '%TH%' or ename like '%LL%';
24. select ename, sal*12 annual_sal, comm from emp where sal > comm order by sal desc, ename;
25. select ename, job, sal, grade, dname from emp, salgrade, dept where emp.deptno = dept.deptno and sal between losal and hisal and job != 'CLERK' order by sal desc;
26. select max(sal) - min(sal) difference from emp;
27. select deptno, dname from dept where deptno not in (select deptno from emp);
28. select deptno, ename, hiredate from emp where (hiredate, deptno) in (select max(hiredate), deptno from emp group by deptno);
29. select ename, sal salary, deptno from emp e where sal > (select avg(sal) from emp where deptno = e.deptno);

30. select job, ename, sal from emp where sal in (select max(sal)
from emp group by job)

31. select ename, job, sal from emp where sal in (select min(sal)
from emp group by job) order by sal desc;

32. select ename, job, sal from emp where job != 'SALESMAN' and
deptno in (select deptno from dept where dname = 'SALES');

33. select job, max(sal) maximum, min(sal) minimum from emp
group by job;

34. select deptno, count(*) from emp group by deptno having
count(*) > 3;

35. select mgr, min(sal) from emp group by mgr having
min(sal) > 1000 order by min(sal);

36. select emps.empno, emps.ename, mgrs.empno mgrno,
mgrs.ename mgr_name from emp emps, emp mgrs where
emps.mgr = mgrs.empno;

37. select emps.empno, emps.ename, mgrs.empno mgrno,
mgrs.ename mgr_name from emp emps, emp mgrs where
emps.mgr = mgrs.empno(+);

38. select d.deptno, dname from emp e, dept d where
e.deptno(+) = d.deptno and e.empno is null;

39. select mgr from emp where deptno = 10 union select mgr from
emp where deptno = 20 order by 1;

40. SELECT job
FROM emp
WHERE hiredate BETWEEN '01/01/81' and '07/01/81'
UNION
SELECT job
FROM emp
WHERE hiredate BETWEEN '01/01/82' and '07/01/82'
ORDER BY job DESC;

41. select e.ename employee, e.hiredate, m.ename manager,
m.hiredate from emp e, emp m where e.mgr = m.empno and
e.hiredate < m.hiredate;

42.

Drop view NONsensempDallas;

CREATE VIEW NONsensempDallas
AS Select empno, ename, mgr, deptno, dname
From emp, dept
Where emp.deptno=dept.deptno
and loc = 'DALLAS'

--basic SQL statement
Select * from NONsensempDallas;


```
SELECT ename  
FROM NONsensempDallas  
Where mgr = 7566;
```

43. All code given. To test or view data in temp table, type `SELECT *`
`FROM temp.`

44. After you have created a trigger, test it.

Type `UPDATE dept SET deptno = 80 Where deptno = 10`

Appendix C: SQL & PL/SQL Quick Reference Guide

The keywords are in UPPERCASE

User defined names are in lower or mixed case

Optional items are enclosed in [] don't include the [] symbols though!

Creating a table

```
CREATE TABLE tablename  
(column/attribute1 datatype [CONSTRAINT constraint_name ] constraint_type...,  
(column/attribute2 datatype [CONSTRAINT constraint_name ] constraint_type...,  
...  
[ CONSTRAINT constraint_name ] constraint_type (column, ...),...);
```

Column-level constraint

```
Column datatype [CONSTRAINT constraint_name ] constraint_type,
```

Table-level constraint

```
[CONSTRAINT constraint_name ] constraint_type (column, ...),
```

Adding a column to an existing table

```
ALTER TABLE tablename ADD columnname datatype;
```

Modifying an existing column

```
ALTER TABLE tablename ADD [CONSTRAINT constraintname] constraint_type  
(columnname/expression)  
[REFERENCES tablename (columnname) ]
```

Dropping a column

```
ALTER TABLE tablename DROP COLUMN columnname;
```

Setting a column as unused

```
ALTER TABLE tablename SET UNUSED (columnname);
```

Dropping an unused column

```
ALTER TABLE tablename DROP UNUSED COLUMNS;
```

Dropping a table

```
DROP TABLE tablename;
```

Renaming a table

```
RENAME oldtablename TO newtablename;
```

Truncating a table (removes all rows but leaves structure)

```
TRUNCATE TABLE tablename;
```

Inserting a new row into a table

```
INSERT INTO tablename [(column1, column2, column3,...)] VALUES (value1, value2, value3,...);
```

Customised prompts

```
ACCEPT variablename PROMPT 'prompt message';
```

Updating row(s)

```
UPDATE tablename SET column1 = newvalue  
[,column2 = newvalue,...] [WHERE condition];
```

Deleting row(s)

```
DELETE [FROM] tablename [ WHERE condition];
```

Dropping a constraint

```
ALTER TABLE tablename DROP PRIMARY KEY /UNIQUE (columnname) /  
CONSTRAINT constraintname [CASCADE];
```

Enabling/disabling a constraint

```
ALTER TABLE tablename DISABLE CONSTRAINT constraintname [CASCADE];  
ALTER TABLE tablename ENABLE CONSTRAINT constraintname;
```

Retrieving data from a table

```
SELECT column, groupfunction (column) FROM tablename  
[WHERE condition(s)] GROUP BY column/expression]  
[ORDER BY column/expression [ASC/DESC]];
```

Define command

```
DEFINE variable [=value]
```

Decode function

```
DECODE (column / expr, value1, action1, [value2, action2, ...] [, default]);
```

Joining tables inner or outer join

```
SELECT tablename1.columnname, tablename2.columnname FROM tablename1, tablename2  
WHERE tablename1.columnname [(+)] = tablename2.columnname [(+)];
```

Set operation

```
SELECT query1 SetOperator SELECT Query2;
```

Select sub-query

```
SELECT columnlist FROM tablename WHERE columnname operator  
(SELECT columnlist FROM tablename WHERE condition);
```

Creating a table using a sub-query

```
CREATE TABLE tablename AS SELECT query;
```

Inserting a row using a sub-query

```
INSERT INTO tablename [(column aliases)]  
SELECT columnnames FROM tablename WHERE condition;
```

Updating using a sub-query

```
UPDATE tablename SET (columnnames) operator  
(SELECT-FROM –WHERE sub-query) WHERE condition;
```

Deleting using a sub-query

```
DELETE FROM tablename WHERE columnname operator  
(SELECT-FROM-WHERE sub-query);
```

Creating a view

```
CREATE [OR REPLACE] [FORCE/NOFORCE] VIEW viewname [column aliases] AS  
SELECT – subquery [WITH CHECK OPTION [CONSTRAINT constraintname]] [WITH  
READ ONLY];
```

Dropping a view

```
DROP VIEW viewname;
```

Creating a sequence

```
CREATE SEQUENCE sequencename  
[INCREMENT BY n] [START WITH s] [ MAXVALUE x / NOMAXVALUE]  
[MINVALUE m / NOMINVALUE] [CYCLE / NOCYCLE] [CACHE c  
/ NOCACHE][ORDER / NOORDER];
```

Modifying a sequence

```
ALTER SEQUENCE sequencename [INCREMENT BY n] [MAXVALUE x /  
NOMAXVALUE] [MINVALUE m / NOMINVALUE] [CYCLE / NOCYCLE] [CACHE /  
NOCACHE] [ORDER / NOORDER];
```

Creating a synonym

```
CREATE [PUBLIC] SYNONYM SynonymName FOR objectname;
```

Creating an index

```
CREATE INDEX indexname ON tablename (columnname1 [columnname2]...);
```

Locking row(s) for update

```
SELECT columnnames FROM tablename WHERE condition  
FOR UPDATE OF columnnames [NOWAIT];
```

Creating a user

```
CREATE USERR username IDENTIFIED BY password;
```

Granting system privileges

```
GRANT privilege 1 [, privelege2...] TO username1 [username2 ...];
```

Granting object privileges

```
GRANT objectpriveleges [(columnnames)] / ALL
```

ON objectname TO user/role/PUBLIC [WITH GRANT OPTION];

Revoking privileges

REVOKE privilege1 [, privilege2...] / ALL on objectname FROM user/role/PUBLIC
[CASCADE CONSTRAINTS];

PL/SQL Anonymous block

DECLARE declaration of constants, variables, cursors and exceptions
BEGIN
 PL/SQL and SQL statements
EXCEPTION
 Actions for error conditions
END;

PL/SQL variable/constant declaration

DECLARE
 IdentifierName [CONSTANT] DataType [NOT NULL] [:=/DEFAULT expression];

Anchored declaration

VariableName TypeAttribute%TYPE [value assignment];

Assignment operation

VariableName := Literal/VariableName/Expression;

IF-THEN-END IF

IF condition THEN Action statements END IF;

IF-THEN-ELSE-END-IF

IF condition THEN actionstatements1 ELSE actionstatements2 END IF

IF-THEN-ELSEIF-END IF

IF condition1 THEN action statements 1
ELSEIF condition2 THEN action statements2
...
ELSEIF condition THEN Action statement N
[ELSE Else Action statements]
END IF

Basic Loop

LOOP
 Looping statement1;
 ...
EXIT [WHEN condition];
END LOOP;

While Loop

WHILE condition LOOP
 Looping statement 1;
 Looping statement 2;
 ...
 looping statement n;

```
END LOOP;
```

For Loop

```
FOR counter IN [REVERSE] lower..upper LOOP
    Looping statement 1
    Looping statement 2
    ...
    Looping statement n
END LOOP;
```

Bind/Host variable

```
VARIABLE VariableName DataType
```

Select into in PL/SQL

```
SELECT ColumnNames INTO VariableNames / RecordName
FROM TableName WHERE condition;
```

Explicit cursor declaration

```
CURSOR CursorName IS SELECT statement;
```

Opening an explicit cursor

```
OPEN CursorName;
```

Fetching a row from an explicit cursor

```
FETCH CursorName INTO VariableList /RecordName;
```

Closing an explicit cursor

```
CLOSE CursorName;
```

Cursor For Loop

```
FOR RecordName IN CursorName LOOP
    Loop statements;
END LOOP;
```

Where current of clause

```
UPDATE TableName SET clause WHERE CURRENT OF CursorName;
```

Cursor with select-for update

```
CURSOR CursorName IS
    SELECT ColumnNames FROM TableName [WHERE condition]
FOR UPDATE [OF ColumnNames] [NOWAIT];
```

Cursor with parameters

```
CURSOR CursorName [(Parameter1 DataType, Parameter 2 DataType, ...)]
IS SELECT query;
```

REF cursor type

```
TYPE CursorTypeName IS REF CURSOR [RETURN ReturnTyp];
CursorVarName CursorTypeName;
Opening a cursor variable
```

```
OPEN CursorName / CursorVarName FOR SELECT query;
```

Fetching from a cursor variable

```
FETCH CursorVarName INTO RecordName / VariableList;
```

Exception section

```
EXCEPTION
WHEN ExceptionName1 [OR ExceptionName2, ...] THEN
    Executable statements
WHEN ExceptionName3 [OR ExceptionName4, ...] THEN
    Executable statements
[WHEN OTHERS THEN Executable statements]
```

Pragma_init_exception directive

```
ExceptionName EXCEPTION;
PRAGMA_EXCEPTION_INIT (ExceptionName, ErrorNumber);
```

Raise_application_error procedure

```
RAISE_APPLICATION_ERROR (error_code, error_message [, TRUE/FALSE]);
```

Creating a PL/SQL Record

```
TYPE RecordTypeName IS RECORD
    (FieldName1 DataType / Variable%TYPE / table.column%TYPE /
    table%ROWTYPE [[NOT NULL] := / DEFAULT Expression]
    [, FieldName2.. , fieldName3...); RecordName RecordTypeName;
```

Declaring a PL/SQL Table

```
TYPE TableTypeName IS TABLE OF
    DataType / VariableName% TYPE / TableName.ColumnName%TYPE
    [NOT NULL] INDEX BY BINARY_INTEGER;
TableName TableTypeName;
```

PL/SQL Procedure

```
CREATE [ OR REPLACE] PROCEDURE ProcedureName
    [(parameter1 [, parameter2...])]
IS
    [Constant/Variable declarations]
BEGIN
    Executable statements
[EXCEPTION
    exception handling statements]
END [ ProcedureName];
```

Calling a procedure

```
ProcedureName [ (parameter1, ...)];
```

Recompiling a procedure

```
ALTER PROCEDURE ProcedureName COMPILE;
```

PL/SQL Function

```
CREATE [ OR REPLACE ] FUNCTION FunctionName  
    [(parameter 1 [, parameter2... ])] RETURN DataType  
IS [Constant / Variable declarations]  
BEGIN executable statements RETURN ReturnValue  
[EXCEPTION exception handling statements RETURN ReturnValue ] END [FunctionName];
```

PL/SQL Package specification

```
CREATE [ OR REPLACE] PACKAGE PackageName  
IS [constant, variable and type declarations ]  
[exception declarations]  
[cursor specifications]  
[function specifications]  
[procedure specifications]  
END [ PackageName];
```

PL/SQL Package body

```
PACKAGE BODY PackageName  
IS  
    [variable and type declarations]  
    [cursor specifications and SELECT queries]  
    [specification and body of functions]  
    [specification and body of procedures]  
[BEGIN executable statements]  
[EXCEPTION exception handlers ]  
END [ PackageName ];
```

PL/SQL Trigger

```
CREATE [ OR REPLACE ] TRIGGER TriggerName  
BEFORE / AFTER TriggeringEvent ON TableName  
[FOR EACH ROW]  
[WHEN condition]  
DECLARE  
    Declaration statements  
BEGIN  
    Executable statements  
EXCEPTION  
Exception handling statements  
END;
```

Creating tablespace

```
CREATE TABLESPACE TablespaceName DATAFILE 'filespecs' SIZE size;
```

Creating a user from the command line using all clauses

```
CREATE USER UserName IDENTIFIED BY PasswordName  
[DEFAULT TABLESPACE DefaultTableSpaceName]  
[TEMPORARY TABLESPACE TemporaryTableSpaceName]  
[QUOTA StorageSpace ON DefaultTableSpaceName]  
[PROFILE ProfileName];
```

Logging into sql*Plus from the command line

SQLPLUS [UserName [/Password]] [@HostName] [@script] [parameter list]

Oracle Web Sites

ORACLE www.oracle.com

ORACLE Technology Network <http://www.oracle.com/technology/index.html>

ORACLE user's group www.oug.com

ORACLE Magazine www.oramag.com

Appendix D: Trouble Shooting Common APEX Oracle Error Messages



For a full reference of all Oracle Server Messages, please visit
http://docs.oracle.com/cd/B10500_01/server.920/a96525/rdbms_pa.htm

If you visit this link: http://docs.oracle.com/cd/B10500_01/server.920/a96525/toc.htm, you will be able to find messages for PL/SQL, Network messages, and much more.

Alternatively visit this page:

http://docs.oracle.com/cd/E11882_01/server.112/e17766/toc.htm

Summary

This document is intended to familiarise you with common ORACLE errors. In particular it should indicate where you have gone wrong and give you an idea of how to fix the problem.

This document looks at the following:

Common Error messages when creating or dropping tables

1. Invalid table name (ORA-00903)
2. Invalid column name (ORA-00904)
3. Invalid data type (ORA-00902)
4. Table or view does not exist (ORA-00942)

Common Error messages when populating tables

1. Missing left parenthesis (ORA-00906)
2. Missing right parenthesis (ORA-00907)
3. Too many values (ORA-00913)
4. Missing comma (ORA-00917)
5. Inserted value too large for column (ORA-01401)
6. Value larger than specified precision allows for this column (ORA-01438)
7. Not enough values (ORA-00947)
8. Cannot insert NULL into (ORA-01400)
9. Integrity constraint (ORA-02291)
10. Unique constraint (ORA-00001)
11. Quoted string not properly terminated (ORA-01756)

Common Error messages when executing queries

1. Missing Expression (ORA-00936)
2. Column ambiguously defined (ORA-00918)
3. Not a GROUP BY expression (ORA-00979)

Common Error messages when creating or dropping tables

The code can be run as a SQL scripts or in the SQL command tool environment. In both environments you will get the same error message.

1. Invalid table name (ORA-00903)

Oracle returns an error when a table name does not satisfy Oracle object-naming requirements. For example, the first letter of a table name must be a letter.

```
CREATE TABLE 10guest(  
    guest_number NUMBER(5) PRIMARY KEY,  
    first_name VARCHAR2(30))
```

ORA-00903: invalid table name

Cause: A table or cluster name is invalid or does not exist. This message is also issued if an invalid cluster name or no cluster name is specified in an ALTER CLUSTER or DROP CLUSTER statement.

Action: Check spelling. A valid table name or cluster name must begin with a letter and may contain only alphanumeric characters and the special characters \$, _, and #. The name must be less than or equal to 30 characters and cannot be a reserved word. Table name in this example should be called guest10 rather than 10 guest.

2. Invalid column name (ORA-00904)

Oracle returns an error when a column name does not satisfy Oracle object-naming requirements. For example, the first letter of a column name must be a letter.

```
CREATE TABLE guest(  
    10guest_number NUMBER(5) PRIMARY KEY,  
    first_name VARCHAR2(10))
```

ORA-00904: invalid identifier

Cause: The column name entered is either missing or invalid.

Action: Enter a valid column name. A valid column name must begin with a letter, be less than or equal to 30 characters, and consist of only alphanumeric characters and the special characters \$, _, and #. If it contains other characters, then it must be enclosed in double quotation marks. It may not be a reserved word. Column name in this case should be called guest_number10 rather than 10guest_number.

Oracle also returns this error if the specified column is not part of the table.

```
SELECT invoice_date FROM guest
```

ORA-00904: "INVOICE_DATE": invalid identifier

3. Invalid data type (ORA-00902)

Oracle returns this error when an invalid data type is referenced in an SQL statement. For example, the following SQL statement attempts to create a column with an illegal data type. (The illegal data type here would be 'TIME')

```
CREATE TABLE guest(  
  guest_number NUMBER(5) PRIMARY KEY,  
  time_of_day TIME)
```

ORA-00902: invalid datatype

Cause: The data type entered in the CREATE or ALTER TABLE statement is not valid.

Action: Correct the syntax. The correct data type here would be TIMESTAMP.

4. Table or view does not exist (ORA-00942)

This common error usually gets people worried. You would come across this error message if trying to drop a table that has never been created. Once the table is created in your user space you are then able to drop it without any errors. This error could also come up when you are referencing a table and you have misspelled the table name.

```
DROP TABLE guest CASCADE CONSTRAINTS
```

ORA-00942: table or view does not exist

```
DROP TABLE guest
```

ORA-00942: table or view does not exist

Cause: The table or view entered does not exist. Existing user tables and views can be listed by querying the data dictionary use `SELECT * FROM CAT`. If an application returned this message, the table the application tried to access does not exist in the database, or the application does not have access to it.

Action: Check each of the following:

- the spelling of the table or view name.
- that a view is not specified where a table is required.
- that an existing table or view name exists.

If attempting to access a table or view in another schema, make certain the correct schema is referenced and that access to the object has been granted.

Common Error messages when populating tables

1. Missing left parenthesis (ORA-00906)

If Oracle doesn't detect the left parenthesis (that it expects in an SQL statement, it returns an error message.

ORA-00906: missing left parenthesis

Cause: A required left parenthesis has been omitted. Certain commands, such as CREATE TABLE, CREATE CLUSTER, and INSERT, require a list of items enclosed in parentheses (). Parentheses also are required around sub queries in WHERE clauses and in UPDATE *table* SET *column* = (SELECT...) statements.

Action: Correct the syntax, inserting a left parenthesis where required, and retry the statement.

2. Missing right parenthesis (ORA-00907)

If Oracle doesn't detect the right parenthesis) that it expects in a SQL statement, it returns an error message.

```
CREATE TABLE invoice(  
    inv_number NUMBER(5) PRIMARY KEY,  
    inv_date VARCHAR2(10),  
    guest_no NUMBER(5) references guest(guest_number)
```

ORA-00907: missing right parenthesis

Cause: A left parenthesis has been entered without a closing right parenthesis, or extra information was contained in the parentheses. All parentheses must be entered in pairs.

Action: Correct the syntax and retry the statement. This example is missing right, closing, bracket.

3. Too many values (ORA-00913)

This error occurs when the number of columns specified in an INSERT statement is less than the number of column values, as shown in this example. (the table guest in this example can only hold two values and gives an error when you try to put in more)

```
INSERT INTO guest VALUES (9, 'Helena','Pat')
```

ORA-00913: too many values

Cause: The SQL statement requires two sets of values equal in number. This error occurs when the second set contains more items than the first set. For example, the sub query in a WHERE or HAVING clause may return too many columns, or a VALUES or SELECT clause may return more columns than were listed in the INSERT.

Action: Check the number of items in each set and change the SQL statement to make them equal. In this example 'Pat' is not an included value.

4. Missing comma (ORA-00917)

If Oracle is expecting a comma in an SQL statement that doesn't have one, you get the following error message. (In this example there should be a comma after the number '9')

```
INSERT INTO guest VALUES (9 'Helena')
```

ORA-00917: missing comma

Cause: A required comma has been omitted from a list of columns or values in an INSERT statement or a list of the form ((C,D),(E,F), ...).

Action: Correct the syntax.

5. Inserted value too large for column (ORA-01401 and ORA-12899)

If you try to assign a character string, whether it is a constant or a column expression and its length exceeds the targeted column, Oracle returns an error message. (The second value in this example exceeds the maximum character size.)

```
INSERT INTO guest VALUES (9, 'thisexceedslengthofcharrrrrr')
```

ORA-12899: value too large for column "LAZARE01"."GUEST"."FIRST_NAME" (actual: 28, maximum: 10)

Cause: The value entered is larger than the maximum width defined for the column.

Action: Enter a value smaller or equal to the column width/size. Alternatively use the MODIFY option with ALTER TABLE to expand the column width.

6. Value larger than specified precision allows for this column (ORA-01438)

If you try to assign a number and its length exceeds the targeted column, Oracle returns an error message. (The first value in this example exceeds the maximum character size. In this example the first value is set to 5 characters which it exceeds).

```
INSERT INTO guest VALUES (989654, 'Pat')
```

ORA-01438: value larger than specified precision allowed for this column

Cause: When inserting or updating records, a numeric value was entered that exceeded the precision defined for the column.

Action: Enter a value that complies with the numeric column's precision, or use the MODIFY option with the ALTER TABLE command to expand the precision.

7. Not enough values (ORA-00947)

This error can occur when the number of columns specified in an INSERT statement is greater than the number of column values to be inserted, as shown in the following examples.

```
INSERT INTO guest VALUES (9)
```

ORA-00947: not enough values

Cause: This error occurs when an SQL statement requires two sets of values equal in number, but the second set contains fewer items than the first set. Another common cause of this error is an INSERT statement in which the number of values does not match those in the INSERT.

Action: Check the number of items in each set and change the SQL statement to make them equal. In this case the first name column value is missing, and if we want to insert a blank value, we have to do something like ... VALUES (9, ' ') or VALUES (9, NULL)

8. Cannot insert NULL into (ORA-01400)

This error occurs when you don't supply a value for a mandatory column. (In the example below I am only inserting a value in the two columns identified and trying to avoid putting a value in for guest_number which is the guest table as a primary key.)

```
INSERT INTO guest (first_name) VALUES ('Pat')
```

ORA-01400: cannot insert NULL into ("LAZARE01"."GUEST"."GUEST_NUMBER")

Cause: An attempt was made to insert a NULL into the column "USER"."TABLE"."COLUMN".

Action: Retry the operation with a value other than NULL. In this example, guest_number must have a value.

9. Integrity constraint (ORA-02291)

This error occurs when the foreign key does not match a primary key. In this example I have executed the invoice table which has a foreign key referencing the guest table. Inside the guest table there are two primary key values, 1 and 3. The value '2' in this case does not exist as a primary key value in the referenced table. By putting another row of data in the guest table and setting the value 2 for the primary key this should work.

```
INSERT INTO invoice (inv_number, inv_date, guest_no)
VALUES (1, '12/12/12', 2)
```

ORA-02291: integrity constraint (LAZARE01.SYS_C001351958) violated - parent key not found

Cause: A foreign key value has no matching primary key value.

Action: Delete the foreign key or add a matching primary key. In this example, there is no guest number in guest table of a value 2.

10. Unique constraint (ORA-00001)

This error would occur when you insert a value where the primary key already exists. (In this example the value '1' has already been inserted to the table and by repeating the same value prompts you with an error.)

```
INSERT INTO guest VALUES (1, 'Pat')
```

1 row(s) inserted. 0.02 seconds

```
INSERT INTO guest VALUES (1, 'John')
```

ORA-00001: unique constraint (LAZARE01.SYS_C001351956) violated

11. Quoted string not properly terminated (ORA-01756)

```
INSERT INTO guest VALUES (9, Helena')
```

ORA-01756: quoted string not properly terminated

Cause: A quoted string must be terminated with a single quote mark (').

Action: Insert the missing closing quote and retry the statement.

Common Error messages when executing queries

1. Missing Expression (ORA-00936)

This error occurs when Oracle expects to see an expression. In this query a comma follows the surname column, but no more columns or expressions are in the select list. (This could be corrected by eliminating the last comma)

```
SELECT guest_number, first_name, surname,  
FROM guest
```

ORA-00936: missing expression

Cause: A required part of a clause or expression has been omitted. For example, a SELECT statement may have been entered without a list of columns or expressions or with an incomplete expression. This message is also issued in cases where a reserved word is misused, as in SELECT TABLE.

Action: Check the statement syntax and specify the missing component.

2. Column ambiguously defined (ORA-00918)

This error occurs when a SQL statement references two or more tables. If a column exists in more than one of the specified tables and is referenced in the SQL statement without qualifying the column with its table name, the column is said to be ambiguous. (the solution here would be to use Alias)

```
SELECT guest_number, first_name
FROM guest, invoice
WHERE guest.guest_number = invoice.guest_number
```

ORA-00904: "INVOICE"."GUEST_NUMBER": invalid identifier

Cause: A column name used in a join exists in more than one table and is thus referenced ambiguously. In a join, any column name that occurs in more than one of the tables must be prefixed by its table name when referenced. The column should be referenced as TABLE.COLUMN or TABLE_ALIAS.COLUMN. For example, if tables EMP and DEPT are being joined and both contain the column DEPTNO, then all references to DEPTNO should be prefixed with the table name, as in EMP.DEPTNO or E.DEPTNO.

Action: Prefix references to column names that exist in multiple tables with either the table name or a table alias and a period (.), as in the examples above.

3. Not a GROUP BY expression (ORA-00979)

Oracle returns this error if a column in a query's select list is contained in a GROUP BY clause and another column in the select list is not.

```
SELECT guest_number, first_name
FROM guest
GROUP BY guest_number
```

ORA-00979: not a GROUP BY expression

Cause: The GROUP BY clause does not contain all the expressions in the SELECT clause. SELECT expressions that are not included in a group function, such as AVG,

COUNT, MAX, MIN, SUM, STDDEV, or VARIANCE, must be listed in the GROUP BY clause.

Action: Include in the GROUP BY clause all SELECT expressions that are not group function arguments.

In order to correct the above statement you would need to add the column first_name in the GROUP BY list shown in the example below.

```
SELECT guest_number, first_name  
FROM guest  
GROUP BY guest_number, first_name;
```

Appendix E: Apex and SQL Documentation

Introduction to the Oracle APEX environment

Oracle Application Express (APEX) is a rapid application development tool for producing web applications using an ORACLE database.

The applications can be built using a web browser with only limited programming experience, although you will need to be familiar with ORACLE SQL. APEX allows you to quickly develop professional looking applications which are fast and secure.

Please note that this year APEX has been upgraded to version 5. Apart from the user interface being clearer, the majority of features and the development environment have not changed.

Oracle Application Express Components

There are 4 main components:

- **Application Builder** – used to build database web enabled applications.
- **SQL Workshop** – used to access database objects and to run SQL statements or scripts.
- **Team Development** – facilitates the management of the application development process; use this tool to track features, to do tasks, milestones and bugs.
- **Administration** – used to manage services and users and to monitor system activity.

What's new?

Utilities have been moved from Builder to SQL Workshop

Architecture

Oracle Application Express is contained within the ORACLE database and is made up of about 165 tables of data and 300,000 lines of PL/SQL code.

When you run an application your browser sends a URL request which is translated into an Oracle Application Express PL/SQL call. When the database has processed the call the results are displayed back in your browser as HTML. This is repeated each time you request or submit a page. The application session state is managed within Oracle Application Express's tables. Uploading SQL scripts

What are SQL Scripts?

An SQL script is a set of SQL commands saved as a file in SQL Scripts. A SQL script can contain one or more SQL statements or PL/SQL blocks. You can use SQL Scripts to create, edit, view, run, and delete database objects (tables, views and triggers).

The script is parsed during runtime. When parsed, ignored statements such as SQL*PLUS are listed. Any invalid SQL is identified in the results. If a script of the same name exists in the Script Repository, you are prompted to rename it.

Creating tables and inserting data using Object Browser

A table is a unit of data storage in an Oracle database containing rows and columns. Each table must have a table name, column names, data types for each column, a size and any constraints on the table or column that are necessary.

There are a number of different ways to create a table in APEX. A table is an object (so is a view, index etc.). In year one, you were taught how to create a table using the SQL scripts editor, which is outlined above. You can complete following task either the way you've learned previously (using the SQL script environment) or follow the instructions in this section to create a table using the Object Browser.



A constraint allows you to restrict the data that can be entered into a column to ensure that it is valid or that it meets certain criteria.

Six types of integrity constraints are described briefly below. More detail on each type comes in later sections.

- A **NOT NULL constraint** prohibits a database value from being null.
- A **unique constraint** prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.
- A **primary key constraint** combines a NOT NULL constraint and a unique constraint in a single declaration.
-
- A **foreign key constraint** requires values in one table to match values in another table.
- A **check constraint** requires a value in the database to comply with a specified condition.
-
- A REF column by definition references an object in another object type or in a relational table. A **REF constraint** lets you further describe the relationship between the REF column and the object it references. This constraint is not discussed further in this module.

Primary Key

A primary key is a single field or combination of fields that uniquely identifies a record. If you need to define multiple attributes as the key for your table, then choose no primary key at this stage and create your key at a later stage. When defining

multiple attributes as a primary key that is called the **Compound** key, also called a **Composite** key or concatenated key.

Below is an explanation of each alternative option for you to be aware of in Object Brower:

- **No Primary Key** - No primary key (PK) is created in this table. Note that if you choose this option you can create a PK later using the constraints tab in the Object Brower.
-
- **Populate from a new sequence** - Creates a primary key, trigger and a sequence. The sequence is used in the trigger to populate the selected primary key column. The primary key can only be a single column.
- **Populated from an existing sequence** – Automatically creates a sequence for a primary key and creates a trigger. The selected sequence is used in the trigger to populate the selected primary key column. The primary key can only be a single column.
- **Not populated** - Defines a primary key, but does not automatically populate the key with a sequence within a trigger. You can also select this option to define a composite primary key (a primary key made up of multiple columns).
- **NEW to Apex 5: Populate By Identity Column** - Creates a primary key and a sequence. The sequence is used to populate the selected primary key column.

Creating a Sequence

To create a sequence:

- On the Workspace home page, click **SQL Workshop** and then **Object Browser**. Object Browser appears.
- Click **Create**. From the list of object types, select **Sequence**.
- For Define: Sequence Name - Enter the name of the sequence.
- Preserve Case - To have the final sequence name match the case entered in the Sequence Name field, click **Preserve Case**.
- Start With - Enter the number of the first sequence. The first reference to sequence_name.nextval returns this number.
 - Minimum Value - Enter the minimum value this sequence can return.
 - Maximum Value - Enter the maximum value this sequence can return.
 - Increment By - Each call to sequence_name.nextval returns a value greater than the last, until the maximum value is reached. Enter the value used to increment to the next sequence number.
 - Cycle - Select this option to restart the sequence number to the minimum value when the maximum value is reached. **This is not recommended if using the sequence for primary key creation.**
- Number to Cache - For faster access, specify how many sequence values are stored in memory.
- Order - Specify **ORDER** to guarantee that sequence numbers are generated in order of request. This option is necessary if using Real Application Clusters (Oracle RAC).
- Click **Next**.
- A confirmation page appears, which displays the SQL used to create the sequence. Click **Create Sequence**.

Browsing a Sequence

- To browse a sequence: On the Workspace home page, click **SQL Workshop** and then **Object Browser**. Object Browser appears.
- From the Object list, select **Sequences**.
- From the Object Selection pane, select a sequence.
- The Object Details view appears. Click the tabs at the top of the page to view different reports about the sequence.

Foreign key

A foreign key (FK) establishes a relationship between a column (or columns) in one table and a primary or unique key in another table.

If your table did require a FK use the following steps to add a foreign key in **Object Browser**.

- a. Name - Enter a name for the foreign key constraint you are defining. Eg. deptno in EMPLOYEE table. This column has to have same Data type and Size defined in EMPLOYEE table as in the PK column in DEPARTMENT table.
- b. Select Key Column(s) - Select the columns that will be part of the foreign key and then click the **Add** icon to move them to Key Column(s).
- c. References Table - Select the table referenced by this foreign key. Then select the columns referenced by this foreign key. Once selected, click the **Add** icon to move the selected columns to Referenced Column(s).
- d. Select **Cascade Delete**.

For an explanation of the other options such as 'Disallow Delete' and 'Set to Null on Delete' see below.

- **Cascade Delete** - Deletes the dependent rows from this table when the corresponding parent table row is deleted.
- **Disallow Delete** - Stops the deletion of rows from the referenced table when there are dependent rows in this table.
- **Set to Null on Delete** - Sets the foreign key column values in this table to null when the corresponding parent table row is deleted.

Example of creating a FK in SQL script

```
CREATE TABLE EMPLOYEE (  
  EMPNO  NUMBER (4) NOT NULL,  
  ENAME  VARCHAR2 (10),  
  JOB    VARCHAR2 (9),  
  MGR    NUMBER (4),
```



```

HIREDATE VARCHAR2 (12),
SAL      NUMBER (7,2),
COMM     NUMBER (7,2),
DEPTNO  NUMBER (2) REFERENCES department (dept_id),
CONSTRAINT PK_EMP PRIMARY KEY ( EMPNO ));

```

ALTER Table FK

You can use ALTER TABLE in the SQL script to add FK for all tables, after your tables have been created with their PK(s). Using ALTER TABLE command EXAMPLE.

```

ALTER TABLE employee ADD CONSTRAINT fk1_deptno FOREIGN KEY(dentno)
REFERENCES department(dept_id);

```

UNIQUE constraint

A **unique constraint** designates a column or a combination of columns as a unique column. To satisfy a unique constraint, no two rows in the table can have the same values for the specified columns.

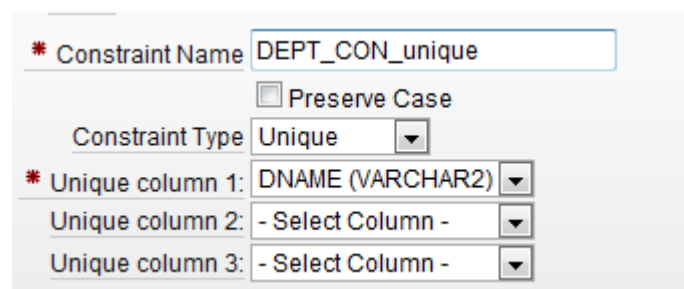
Example 1: Writing within SQL code

```

CREATE TABLE DEPARTMENT (
dept_id NUMBER (3) NOT NULL,
dept_name VARCHAR2 (10) UNIQUE,
loc VARCHAR2 (10),
phone_number VARCHAR2 (15),
CONSTRAINT PK_DEPT PRIMARY KEY (dept_id)

```

Example 2: Defining in Object Browser, Constraint Tab.



The screenshot shows the 'Constraint Tab' in the Oracle Object Browser. The 'Constraint Name' is 'DEPT_CON_unique'. The 'Constraint Type' is 'Unique'. The 'Unique column 1' is 'DNAME (VARCHAR2)'. The 'Unique column 2' and 'Unique column 3' are both set to '- Select Column -'. The 'Preserve Case' checkbox is unchecked.

CHECK constraint

A **check constraint** is a validation check on one or more columns within the table. No records can be inserted or updated in a table which violate an enabled check constraint. Example of check constraint use are below:

```

STATUS IN ('OPEN','CLOSED','PENDING')

```

The view in Object Browser:

Constraint Name	Type	Column(s)/Check
STATUS_CHECK_CONSTRAINT	Check	OPEN','CLOSED','PENDING' X

Constraint Type ☒ Check ☐ Unique

Check Condition

SAL IS NOT NULL OR COMM IS NOT NULL
 LENGTH(ENAME) >= 6
 SAL > 0
 SAL + NVL(COMM,0) > 1000
 ENAME = UPPER(ENAME)
 (SAL * 100) = trunc(SAL * 100)
 HIREDATE > SYSDATE - 7

Editing a Table in Object Browser

While viewing a table description, you can edit it by clicking the buttons above the table description.

To edit a table:

1. On the Workspace home page, click **SQL Workshop** and then **Object Browser**.
Object Browser appears.
From the Object list, ensure **Tables** is selected.
2. From the Object Selection pane, select a table.
The table description appears.

Edit Table Buttons

Button	Description
Add Column	Adds a column to the table. Enter a column name and select a type. Depending upon the column type, specify whether the column requires a value, the column length, precision, and scale.
Modify Column	Modifies the selected column.
Rename Column	Renames the selected column.
Drop Column	Drops the selected column.
Rename	Renames the selected table.
Copy	Copies the selected table.
Drop	Drops the selected table.
Truncate	Removes all rows from the selected table. Truncating a table can be more efficient than dropping and re-creating a table. Dropping and re-creating a table may invalidate dependent objects, requiring you to grant object privileges or re-create indexes, integrity constraints and triggers.

Button	Description
Create Lookup Table	Creates a lookup table based on the column you select. That column becomes a foreign key to the lookup table.

Browsing a Table

When you view a table in the Object Browser the table description appears. While viewing this description, you can add a column, modify a column, rename a column, drop a column, rename the table, copy the table, drop the table, truncate the table, or create a lookup table based upon a column in the current table. Additionally, you have access to other reports that offer related information including the table data, indexes, data model, constraints, grants, statistics, user interface defaults, triggers, dependencies, and SQL to produce the selected table.

To view a table description:

1. On the Workspace home page, click **SQL Workshop** and then **Object Browser**.
Object Browser appears.
2. From the Object list, ensure **Tables** is selected.
3. From the Object Selection pane, select a table.
The table description appears.



How to create a Composite/Compound Primary Key

In the **Object Browser**, select the table for which you want to create a multiple attributes PK. In the tab menu, select **Constraints**, then click **Create**. You will first have to define what constraint you want to create, in this case it's PK, then define which attributes are PK columns in this table. Note that **Constraint Name** has to be unique.

CD_TRACK

Add Constraint

Schema: LAZARE01

Table: CD_TRACK

* Constraint Name: CD_TRACK_PK_Const

☐ Preserve Case

Constraint Type: Primary Key

* Primary Key Column 1: TRACK_CD_IDNO (NUMBER)

Primary Key Column 2: TRACK_NO (NUMBER)

Primary Key Column 3: - Select Column -

Cancel Next >





How to create a Composite/Compound Foreign Key

In the **Object Browser**, select the table for which you want to create multiple attributes as a FK. In the following demonstration I will be using a table called Task. You may not have this table in your account.

From the tab, select **Constraints** and then click **Create**. You will first have to define a **Constraint Name**, set with a default name, but note that it has to be unique for the whole database (data dictionary). Next, define what **Constraint Type** you want to create, in this case, it's a **Foreign Key**, and then select which attributes are FK columns. To select multiple columns, press Ctrl and select all your FK columns. Next, select a PK table and the keys of the table that the TASK table is referenced to. In this case, it's the CD_TRACK table.

Schema: LAZARE01

Table: CD_TRACK

Constraint Name: CD_TRACK_COMPOSITE_FK

Preserve Case: ☐

Constraint Type: Foreign Key

Disallow Delete: ☐ Cascade Delete: ☒ Set Null on Delete: ☐

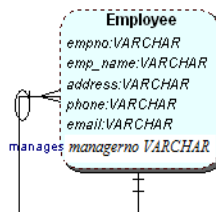
Foreign Key Column(s): TRACK_TITLE, TRACK_LENGTH

Reference Table Name: CD

Reference Table Column(s): CD_ARTIST, CD_PRICE



Creating a table which has a recursive relationship



A recursive relationship is one that relates a table back to itself via an attribute. A PK from one table is implemented in the same table as a FK. For example, empno is the

PK and manager now is the FK (this column has been renamed to make it more meaningful, we could have called it empno_mrg. Values in the manager number column are actual employee IDs (values from empno column.

To add the FK constraint for the managerno column, click on the tab **Constraints** and click on **Create**.

Choose these options to add a foreign key:

- Constraint Name(must be unique) enter: **manager_FK2**
- Select Constraint Type **Foreign Key**
- Select **Cascade Delete** option (deletes the dependent rows from this table when the corresponding parent table row is deleted)
- Foreign Key Columns : Select **managerno**
- References Table Name - Select the table which will be referenced by this foreign key. It is **EMPLOYEE** in this case.
- Then select the column to be referenced by this foreign key, it is **empno**

[note: you normally give FK column, different name to avoid having two columns in one table with the same name and to simplify writing SQL statements later on.]

3. Click **Next** then Finish.

Note: You could have also used ALTER TABLE command, and run the following script in SQL Commands:

```
ALTER TABLE employee ADD CONSTRAINT fk_manager FOREIGN KEY (managerno) REFERENCES emp(empno);
```

SQL Commands Environment

SQL is the standard language used to create and query Relational Databases. In the last section you looked at how to create tables, drop them, insert some data and modify data and columns. This part of the document will focus on the SQL required to do basic and intermediate queries on the **scott** database tables.

Datasheet for the SCOTT database (sample Appendix A) used throughout this workbook in order to help you carry out some of the exercises. You will need to upload this script and run it. **The script can be found on H drive/Slazarevski and on MyBeckett in SQL scripts folder.**

What is an SQL Command window?

You can use the SQL Commands tool window to create, edit, view, run, and delete database objects. An SQL command can contain SQL statements or PL/SQL blocks.

When using SQL Commands, remember the following:

- Saved SQL commands must have unique names within a given workspace.
- There is no interaction between SQL Commands and SQL Scripts.
- You can cut and paste an SQL command from SQL Commands to run in the SQL Script Editor and you can do the same other way around.

Accessing SQL Commands

To access SQL Commands:

1. Log in to the Workspace **Home** page. The Workspace home page appears.
2. To view the SQL Commands home page you can either:
 - Click **SQL Workshop** and then **SQL Commands** to drill-down to the SQL Commands home page.

About the SQL Commands Home Page

The SQL Commands home page is divided into two sections: a command editor and a display pane. You use the command editor to execute an SQL command and the display pane to view the output, saved command lists, and history lists.

The top of the SQL Commands home page features a command editor and the following controls:

- **Autocommit.** If available, click the **Autocommit** check box to enable autocommit and disable transactional commands.
- **Rows.** Select the number of rows of output to display simultaneously up to a maximum of 100,000. All rows of DBMS Output are displayed regardless of the Display list setting.
- **Clear Command.** Use the Clear Command button to clear the text in the command editor.
- **Find Tables.** Use the Find Tables button to view tables within the currently

- selected schema.
- **Save.** Click the **Save** button to save the contents of the command editor, or the currently highlighted content to a file. You are prompted to enter a name and an optional description.
- **Run.** Click the **Run** button (**Ctrl+Enter**) to run the command in the command editor, or the currently highlighted command in the command editor.
- The display pane is at the bottom of the SQL Commands home page and features the following five tabs:
 - **Results.** Click the **Results** tab to see the results from the last successfully executed SQL command. Click **DBMS Output** at the bottom of the displayed results to
- **Explain.** Click the **Explain** tab to examine the execution plan used by the optimizer for statements that make changes to the database. Objects in the output are linked to the Object Browser. Click the linked object to view its properties in the Object Browser.
- **Describe.** Enter *Describe object_name* and click **Run** to display column definitions for a table or view, or specifications for a function or procedure in the **Describe** tab.
- **Saved SQL.** Click the **Saved SQL** tab to display a list of all SQL commands saved in the current workspace.
- **History.** Click the **History** tab to list your recently executed commands. Your last 200 executed commands are saved.

Switching to Another SQL Workshop Component

You can navigate to another SQL Workshop tool by selecting one of the following from the Component list located on the upper right side of the page:

- Object Browser.
- SQL Commands.
- SQL Scripts.
- Query Builder.

About the Display Pane

A display pane displays at the bottom of the SQL Commands home page.

Results Explain Describe Saved SQL History							
CD_IDNO	CD_TITLE	CD_ARTIST	CD_DATE_PURCHASED	CD_PAYMENT_TYPE	CD_MUSIC_CATEGORY	CD_PRICE	COMPANY_ID
1	Live at Leeds	The Who	19-MAR-99	Cash	Rock	10.99	001
2	CLUBLAND 4 THE NIGHT OF YOUR LIFE	VARIOUS	19-JAN-04	Cash	Club	13.99	001
3	CLUBBERS GUIDE 2004	VARIOUS	19-JAN-04	Credit	Club	13.99	001
4	THE VERY BEST OF STING AND THE POLICE	The POLICE	19-MAR-99	Cash	Rock	10.99	002
5	THE RISING	BRUCE SPRINGSTEIN	19-MAR-99	Cash	Rock	10.99	002
6	ROLLING STONES 4 LICKS	The ROLLING STONES	19-MAR-99	Credit	Pop	10.99	003
7	EUPHORIA OLD SCHOOL BREAKDOWN	VARIOUS	19-MAR-99	Cash	Club	10.99	004
8	BEST OF BOWIE	DAVID BOWIE	19-MAR-99	Cash	Pop	10.99	004
9	THE VERY BEST OF EUPHORIA	VARIOUS	19-MAR-99	Cash	Club	10.99	004
10	PINK MISSUNDAZTOOD	PINK	19-MAR-99	Credit	Pop	.99	004

The display pane features five tabs:

- **Results (see above).** Click the **Results** tab to see the results from the last successfully executed SQL command. Click **DBMS Output** on the bottom of the displayed results to display lines of DBMS output. This control only appears when there is DBMS output to display. Click **Download** to export results to a comma-separated file on your local file system. Can be viewed using excel spread sheet.
- **Explain.** Click the **Explain** tab to examine the execution plan used by the optimizer for statements that make changes to the database (Done more in year 3) Objects in the output are linked to the Object Browser. Click the linked object to view its properties in the Object Browser.

Results **Explain** Describe Saved SQL History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			16	1	3	1,088		
TABLE ACCESS	FULL	CD	16	1	3	1,088		

* Unindexed columns are shown in red

Index Columns

Owner	Table Name	Index Name	Used in Plan	Columns	Uniqueness	Status	Index Type	Join Index
SANELA	CD	SYS_C002564141		CD_IDNO	UNIQUE	VALID	NORMAL	NO

Table Columns

Table Owner	Table Name	Column Name	Data Type
SANELA	CD	CD_IDNO	CHAR
		CD_TITLE	VARCHAR2
		CD_ARTIST	VARCHAR2
		CD_DATE_PURCHASED	DATE
		CD_PAYMENT_TYPE	VARCHAR2
		CD_MUSIC_CATEGORY	VARCHAR2
		CD_PRICE	NUMBER
		COMPANY_ID	VARCHAR2

- **Describe.** Enter Describe *object_name* and click **Run** to display column definitions for a table or view, or specifications for a function or procedure (*not covered in this module – see help if you want to know more about these*) in the **Describe** tab. Select links in the Describe results to write that information into the command editor. For example, click a table name to add *owner.table*, click a column name to add the *column name*.
- **Saved SQL.** Click the **Saved SQL** tab to display a list of all SQL commands saved in the current workspace. Click the command title to load it into the command editor.

ORACLE Application Express					Application Builder	SQL Workshop	Team Development	Packaged Apps	LAZARE01
SQL Commands									
Autocommit Rows 15 Clear Command Find Tables Save Run									
select * from emp;									
Results Explain Describe Saved SQL History									
Owner Find Rows 10 Go Delete Checked									
Owner Name Description SQL Updated By Updated									
LAZARE01	select ALL	-	select * from emp;	LAZARE01	Now				

- **History.** Click the **History** tab to list your recently executed commands. Your last 200 executed commands are saved.

Using the SQL Command Editor

About Unsupported SQL*Plus Commands

SQL Command Tool does not support SQL*Plus commands. If you attempt to enter an SQL Command Line command such as SET ECHO or DEFINE in SQL Command Tool, an error message displays.

About Command Termination

You can terminate a command in SQL Commands using a semicolon (;), a forward slash (/), or with nothing. Consider the following valid alternatives:

```
SELECT * from emp;  
SELECT * from emp  
/  
SELECT * from emp
```

The first example demonstrates the use of a semicolon (;), the second example demonstrates the use of forward slash (/), and the final example demonstrates a command with no termination.



Copying/Saving a Command

To copy an SQL command:

1. On the Workspace home page, click **SQL Workshop** and then **SQL Commands**. The SQL Commands page appears.
2. Click the **Saved SQL** tab. The Saved SQL list of commands appears in the display pane.
3. Click the title of the command to load it into the command editor
4. Click **Save** to save the command.
5. Enter a new name for the command in the Name field and click **Save**.

The command is copied to the new name.

Retrieving all the data from a single table

Retrieving all the data from a single table is the simplest method of data selection, this is done using the **SELECT * FROM** statement.

The following syntax defines the **SELECT * FROM** statement:

```
SELECT * FROM tablename;
```

Retrieving data from specific columns from a single table (Projection)

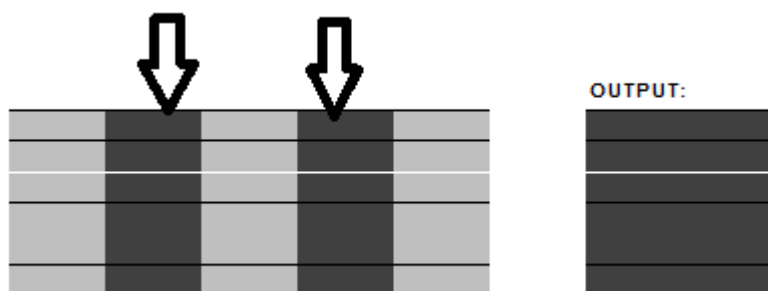
When selecting data from a table, we may not want to select all of the data from the table. Selecting all the data can lead to a very cluttered output, which could include data from the column(s) you are not interested in. In such cases it would be better to only specify the column(s) that you are interested in.

If you want to retrieve one column from a table the syntax is as follows:

```
SELECT columnname  
FROM tablename;
```

If you want to retrieve more than one column from a table the syntax is as follows:

```
SELECT columnname [, columnname, ...]  
FROM tablename;  
[ ... ] content optional
```



Multiple columns are selected by placing the names of the columns to be retrieved in the **SELECT** statement, separated by commas.

Retrieving data from specific rows from a single table (Selection)

When selecting data it is unusual to retrieve every row (or every column of every row) of a large table. More often, rows are retrieved according to a value or range of values in one or more columns of the table. *e.g. You might want details of only the scott category Rock.*

This is achieved by adding the **WHERE** clause to the **SELECT** statement. The **WHERE** clause has the following syntax:

WHERE columnname OPERATOR value;

If you want to retrieve the values of more than one row the syntax is as follows:

WHERE columnname OPERATOR value
AND columnname OPERATOR value
[[AND][OR][NOT] columnname OPERATOR value..];

Character values need to have their values enclosed in single quotes 'hello'.

NOTE: comparison of character values is **case sensitive**. 'A' is not the same as 'a'.

Condition Precedence

Precedence is the order in which Oracle evaluates different conditions in the same expression. When evaluating an expression containing multiple conditions, Oracle evaluates conditions with higher precedence before evaluating those with lower precedence. Oracle evaluates conditions with equal precedence from left to right within an expression.

In Table 1 it lists the levels of precedence among SQL condition from high to low. The conditions listed on the same line have the same precedence. As the table indicates, Oracle evaluates operators before conditions.

Table 1 SQL Condition Precedence

Type of Condition	Purpose
=, !=, <, >, <=, >=,	comparison
IS [NOT] NULL, LIKE, [NOT] BETWEEN, [NOT] IN, EXISTS, IS OF <i>type</i>	comparison

Type of Condition	Purpose
NOT	exponentiation, logical negation
AND	conjunction
OR	disjunction

Comparison Conditions

Comparison conditions compare one expression with another. The result of such a comparison can be TRUE, FALSE, or NULL.

Table 2 Comparison Conditions

Type of Condition	Purpose	Example
=	Equality test.	select ename, job from emp where job = 'CLERK' and deptno = 20; ORDER BY empno;
!=	Inequality test	select ename, job from emp where job! = 'CLERK';
>= <=	Greater-than-or-equal-to and less-than-or-equal-to tests.	SELECT ename, job FROM emp WHERE where sal>= comm;

Join - Selecting data from more than one table

To select the data from more than one table, you list the columns required in the **SELECT** statement, list the tables required in the **FROM** clause, specify the join between the tables (via the Primary and Foreign Keys) in the **WHERE / AND** clauses and set the selection criteria in an additional **AND** statement.

The JOIN operations are:

- **INNER JOIN operation:** Specifies a join between two tables with an explicit join clause.
- **LEFT OUTER JOIN operation:** Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the first table.
- **RIGHT OUTER JOIN operation:** Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the second table.
- **CROSS JOIN operation:** Specifies a join that produces the Cartesian product of two tables. It has no explicit join clause.

- **NATURAL JOIN operation:** Specifies an inner or outer join between two tables. It has no explicit join clause. Instead, one is created implicitly using the common columns from the two tables.

In all cases, you can specify additional restrictions on one or both of the tables being joined in outer join clauses or in the WHERE clause.

The number of join conditions required is the number of tables to be joined minus 1. Therefore, if you wanted to join 3 tables, the number of join conditions would be 2.

The syntax is as follows:

```
SELECT tablename.columnname [,tablename.columnname, ...]
FROM tablename1, tablename2 [,tablename, ...]
WHERE tablename1.primarykey = tablename2.foreignkey
[AND tablename2.primarykey = tablename3.foreignkey, ...]
AND columnname OPERATOR value
[AND columnname OPERATOR value..];
```

The LIKE Condition

The LIKE conditions specify a test involving pattern matching. Whereas the equality operator (=) exactly matches one character value to another, the LIKE conditions match a portion of one character string value to another by searching the first value for the pattern specified by the second. LIKE matches strings using characters as defined by the input character set.

The pattern can contain special pattern-matching characters:

- An underscore (_) in the pattern matches exactly one character (as opposed to one byte in a multibyte character set) in the value.
- A percent sign (%) in the pattern can match zero or more characters (as opposed to bytes in a multibyte character set) in the value. The pattern '%' cannot match a null.

You can include the actual characters % or _ in the pattern by using the ESCAPE clause, which identifies the escape character. If the escape character precedes the character % or _ in the pattern, then Oracle interprets this character literally in the pattern rather than as a special pattern-matching character. You can also search for the escape character itself by repeating it. For example, if @ is the escape character, then you can use @@ to search for @.

Table 3 LIKE Conditions

Type of Condition	Operation	Example
x [NOT] LIKE y [ESCAPE	TRUE if x does [not] match the pattern y. Within y, the character % matches any string of zero or more characters except null. The character _	SELECT* FROM emp WHERE ename

Type of Condition	Operation	Example
'z']	matches any single character. Any character can follow ESCAPE except percent (%) and underbar (_). A wildcard character is treated as a literal if preceded by the escape character.	LIKE 'KI%'; SELECT* FROM emp WHERE ename LIKE '___N%';

To process the LIKE conditions, Oracle divides the pattern into subpatterns consisting of one or two characters each. The two-character subpatterns begin with the escape character and the other character is %, or __, or the escape character.

LIKE Condition: General Examples

This condition is true for all last_name values beginning with Ma:

- last_name LIKE 'Ma%'

All of these last_name values make the condition true:

- Mallin, Markle, Marlow, Marvins, Marvis, Matos

Case is significant, so last_name values beginning with MA, ma, and mA make the condition false.

Consider this condition:

- last_name LIKE 'SMITH_'

This condition is true for these last_name values:

- SMITHE, SMITHY, SMITHS

This condition is false for SMITH because the special underscore character (_) must match exactly one character of the last_name value.

Table 4 Range Conditions

Type of Condition	Operation	Example
[NOT] BETWEEN x AND y	[Not] greater than or equal to x and less than or equal to y.	SELECT * FROM emp WHERE sal BETWEEN 1000 and 1500 ORDER BY empno;

Null Conditions

A NULL condition tests for nulls. This is the only condition that you should use to test for nulls.

Table 5 Null Conditions

Type of Condition	Operation	Example
IS [NOT] NULL	Tests for nulls.	SELECT ename FROM emp WHERE sal IS NOT NULL AND job = 'CLARK';

Using Subqueries

A **subquery** answers multiple-part questions. For example, to determine who works in Taylor's department, you can first use a subquery to determine the department in which Taylor works. You can then answer the original question with the parent SELECT statement. A subquery in the FROM clause of a SELECT statement is also called an **inline view**. A subquery in the WHERE clause of a SELECT statement is also called a **nested subquery**.

A subquery can contain another subquery. Oracle Database imposes no limit on the number of subquery levels in the FROM clause of the top-level query. You can nest up to 255 levels of subqueries in the WHERE clause.

If columns in a subquery have the same name as columns in the containing statement, then you must prefix any reference to the column of the table from the containing statement with the table name or alias. To make your statements easier to read, always qualify the columns in a subquery with the name or alias of the table or a view.

EXISTS Condition

The EXISTS condition tests for the existence of rows in a subquery.

Table 6 EXISTS Condition

Type Condition	of Operation	Example
EXISTS	TRUE if a subquery returns at least one row.	SELECT deptno FROM dept WHERE EXISTS (SELECT * FROM emp WHERE emp.deptno = dept.deptno) ORDER BY deptno;

IN Condition

An *in_condition* is a membership condition. It tests a value for membership in a list of values or a subquery

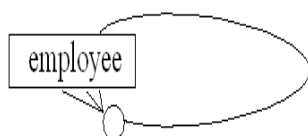
Table 7 IN Conditions

Type Condition	of Operation	Example
IN	Equal-to-any-member-of test. Equivalent to =ANY.	select * from emp where job IN ('CLERK', 'PRESIDENT');
NOT IN	Equivalent to !=ALL. Evaluates to FALSE if any member of the set is NULL.	select * from emp where job NOT IN ('CLERK', 'PRESIDENT');

Using Self-Joins



For this example, we'll have to use emp/dept tables, as our example table does not have a recursive relationship on the model.



EMPLOYEE

TableDataIndexesModelConstraintsGrantsStatisticsUI DefaultsTriggersDependenciesSQL

Add ColumnModify ColumnRename ColumnDrop ColumnRenameCopyDropTruncateCreate Lookup Table

Column Name	Data Type	Nullable	Default	Primary Key
EMP_NO	VARCHAR2(4)	No	-	1
EMP_NAME	VARCHAR2(10)	No	-	-
ADDRESS	VARCHAR2(8)	No	-	-
PHONE_NUMBERS	VARCHAR2(8)	Yes	-	-
EMAIL	VARCHAR2(8)	Yes	-	-
MANAGER_NO	VARCHAR2(4)	Yes	-	-
				1 - 6

Note: Aliases is an alternative column or table name. Aliases are used to make a column name more descriptive, to shorten the column name, or prevent possible ambiguous references.

Example: The following query uses a self join to return the name of each employee along with the name of the employee's manager. A WHERE clause is added to restrict the output.

☒ Autocommit
 Rows
Save Run

```

SELECT e1.emp_name || ' works for ' || e2.emp_name "Employees and Their Managers"
  FROM employee e1, employee e2
 WHERE e1.manager_no = e2.emp_no
 ORDER BY e1.emp_name;
    
```

Results Explain Describe Saved SQL History

Employees and Their Managers	
JONES	works for SMITH
SANELA	works for SMITH
WARD	works for SMITH

3 rows returned in 0.01 seconds [Download](#)

The join condition for this query uses the aliases e1 and e2 for the sample table employees: **e1.manager_no = e2.emp_no**

Group By/Having

When using aggregate functions (MAX, AVG, MIN) that returns a single result row based on groups of rows, rather than on single rows. They are commonly used with the GROUP BY clause in a SELECT statement, where Oracle Database divides the rows of a queried table or view into groups. In a query containing a GROUP BY clause, the elements of the select list can be aggregate functions, GROUP BY expressions, constants, or expressions involving one of these. If you omit the GROUP BY clause, then Oracle applies aggregate functions in the select list to all the rows in the queried table or view.

You use aggregate functions in the HAVING clause to eliminate groups from the output based on the results of the aggregate functions, rather than on the values of the individual rows of the queried table or view.



To list which departments employ more than 10 people we would write this statement:

```
SELECT dname, count(empno)
FROM emp, dept
WHERE emp.deptno = dept.deptno
GROUP BY dname;
```

DNAME	COUNT(EMPNO)
RESEARCH	5
SALES	6
ACCOUNTING	3

If we want a list of those department that have more than 5 employees THEN we will need to add the restriction:

```
SELECT dname, count(empno)
FROM emp, dept
WHERE emp.deptno = dept.deptno
GROUP BY dname
HAVING count(empno) > 5;
```

DNAME	COUNT(EMPNO)
SALES	6

We have to use HAVING count(empno) > 5 when using aggregate function in the condition part of the statement.

Outer Joins

An **outer join** extends the result of a simple join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

- To write a query that performs an outer join of tables A and B and returns all rows from A (a **left outer join**), use the LEFT [OUTER] JOIN syntax in the FROM clause, or apply the outer join operator (+) to all columns of B in the join condition in the WHERE clause. For all rows in A that have no matching

rows in B, Oracle Database returns null for any select list expressions containing columns of B.

- To write a query that performs an outer join of tables A and B and returns all rows from B (a **right outer join**), use the RIGHT [OUTER] JOIN syntax in the FROM clause, or apply the outer join operator (+) to all columns of A in the join condition in the WHERE clause. For all rows in B that have no matching rows in A, Oracle returns null for any select list expressions containing columns of A.
- To write a query that performs an outer join and returns all rows from A and B, extended with nulls if they do not satisfy the join condition (a **full outer join**), use the FULL [OUTER] JOIN syntax in the FROM clause.

You cannot compare a column with a subquery in the WHERE clause of any outer join, regardless which form you specify.

Outer join queries that use the Oracle join operator (+) are subject to the following rules and restrictions, which do not apply to the FROM clause OUTER JOIN syntax:

- You cannot specify the (+) operator in a query block that also contains FROM clause join syntax.
- The (+) operator can appear only in the WHERE clause or, in the context of left-correlation (when specifying the TABLE clause) in the FROM clause, and can be applied only to a column of a table or view.
- If A and B are joined by multiple join conditions, then you must use the (+) operator in all of these conditions. If you do not, then Oracle Database will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.
- The (+) operator does not produce an outer join if you specify one table in the outer query and the other table in an inner query.
- You cannot use the (+) operator to outer-join a table to itself, although self joins are valid. For example, the following statement is **not** valid:

```
-- The following statement is not valid:  
SELECT employee_id, manager_id  
FROM employees  
WHERE employees.manager_id(+) = employees.employee_id;
```

However, the following self join is valid:

```
SELECT e1.employee_id, e1.manager_id, e2.employee_id  
FROM employees e1, employees e2  
WHERE e1.manager_id(+) = e2.employee_id  
ORDER BY e1.employee_id, e1.manager_id, e2.employee_id;
```

- The (+) operator can be applied only to a column, not an arbitrary expression. However, an arbitrary expression can contain one or more columns marked with the (+) operator.
- A WHERE condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A WHERE condition cannot use the IN comparison condition to compare a column marked with the (+) operator with an expression.



The query below will display the department that has no employees.

```
select d.deptno, dname
from    emp e, dept d
where   e.deptno(+) = d.deptno
and     e.empno is null;
```



If you would want to list all employees by name and number along with their manager's name and number, your query would be like:

```
select emps.empno,
       emps.ename,
       mgrs.empno mgrno,
       mgrs.ename mgr_name
from    emp emps, emp mgrs
where   emps.mgr = mgrs.empno;
```

However, to display KING who has no manager, you will have to use the OUTER JOIN expression.

```
select emps.empno,
       emps.ename,
       mgrs.empno mgrno,
       mgrs.ename mgr_name
from    emp emps, emp mgrs
where   emps.mgr = mgrs.empno(+);
```

Table Expression

A Table Expression specifies a table, view, or function in a FROM clause. It is the source from which a Select Expression selects a result.

A correlation name can be applied to a table in a Table Expression so that its columns can be qualified with that name. If you do not supply a correlation name, the table name qualifies the column name. When you are given a table a correlation

(also known as table alias) name, you cannot use the table name to qualify columns. You must use the correlation name when qualifying column names.

No two items in the FROM clause can have the same correlation name, and no correlation name can be the same as an unqualified table name specified in that FROM clause.

In addition, you can give the columns of the table new names in the AS clause. Some situations in which this is useful:

- When a VALUES expression is used as a Table Subquery, since there is no other way to name the columns of a VALUES expression.
- When column names would otherwise be the same as those of columns in other tables; renaming them means you don't have to qualify them.

The Query in a Table Subquery appearing in a From Item can contain multiple columns and return multiple rows. See TableSubquery.



-- SELECT from a JOIN expression

```
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E LEFT OUTER JOIN
    DEPARTMENT INNER JOIN EMPLOYEE M
ON MGRNO = M.EMPNO
ON E.WORKDEPT = DEPTNO
```

VIEWS

A view is a logical representation of another table or combination of tables. A view does not contain or store data but derives its data from the views or tables on which it is based. These tables are called **base tables**. All operations performed on a view affect the base table of the view. In order to be updatable, a view cannot contain any of the following constructs: SET or DISTINCT, operators, aggregate or analytic functions, GROUP BY, ORDER BY, CONNECT BY, START WITH clause, subquery, or collection expression in a SELECT list. For an example where using a view might be preferable to a table, consider the HR.EMPLOYEES table which has several columns and numerous rows.

Please check separate tutorials on VIEWS available on the Modules VLE.

Document End ☺