STRUCTURED QUERY LANGUAGE

INFO503 – Database Principles

Compiled by:
Sunitha Prabhu
Centre for Business, Information Technology and Enterprise (CBITE)
Waikato Institute of Technology (Wintec)

Table of Contents

LAB SESSION 1: CREATE TABLES 4
- Types of SQL 4
- Tables 5
- Constraints 6
- Data Types 7
- SQL Syntax 8
- Writing a script 9
- Creating a table 10
- Running a script 11
- Help from system tables 11
- Foreign Key Constraints 14

LAB SESSION 2: POPULATING TABLES 15
- Updating Rows 21
- Updating Selected Rows 21
- Deleting all records 22
- EXERCISE 2 22

LAB SESSION 3: RETRIEVING DATA FROM A TABLE 24
- Projection 24
- Operators 25
- Using AND, OR, NOT clause 25
- EXERCISE 3 26
- Column Headings 27

LAB SESSION 4: SELECT statements 29
- Using LIKE clause 29
- Aggregate functions 30
- EXERCISE 4 31

LAB SESSION 5: JOINS 33
- Selection from Multiple Tables 33
- Using VIEW 35
- EXERCISE 5 36

I THINK WE SHOULD BUILD AN SQL DATABASE.
UH-OH
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
WHAT COLOR DO YOU WANT THAT DATABASE?
I THINK MAUVE HAS THE MOST RAM.

Welcome to the SQL component of INFO503 ☺

The SQL component of your INFO503 – Database Principles course is learned in lab sessions. Each lab session is clearly shown in this booklet and it is up to each student to complete the exercises at the end of each session before going on to the next session. Some of you will do this easily within the lab times; others will need to come back in to do extra work. In all cases it is up to the individual student to understand the SQL principles given in each session and most students will need to study outside of the classroom to achieve this. Your tutor will structure the lab sessions and answer your questions and to look at your completed exercises.

This part of the course is assessed by one test: a mixture of theory and practical. All of the course work necessary to pass the SQL test is available in this workbook.

This workbook will cover the following component of INFO503.

3. Structured Query Language (SQL)

- Data definition
    - Create database objects (tables and views)
    - Drop database objects
- Data manipulation
    - Insert data
    - Update data
    - Delete data
- Data integrity constraints
    - Primary key
    - NULL
    - Foreign key
    - Default
- Querying and Reporting
    - Select records from tables
    - Conditional Select
    - Simple Joins
    - Format Output

LAB SESSION 1: CREATE TABLES

Each DBMS can have a variety of methods to access the data held, but rather than each vendor inventing a new approach, standards do exist. These languages are often called *Data Sub-Languages* (DSL), and are really a combination of two languages; a *Data Definition Language* (DDL) which provides for the description of database objects and a *Data Manipulation Language* (DML) which supports the manipulation or processing of such objects.

SQL is a standard language for accessing and communications with a database. SQL was originally developed in IBM in the early 1970s. It stands for *Structured Query Language* and is pronounced either *ess-kew-ell* or *sequel*. SQL is a fourth generation language and as such does not contain some traditional programming commands such as loops and branches.

Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, DELETE, INSERT, UPDATE, DELETE, WHERE) in a similar manner. SQL has some variations such as ISO SQL, ORACLE SQL, T-SQL, SYBASE SQL, MS ACCESS.

Types of SQL

SQL statements can be divided into three main types:
- Data Definition Language statements (DDL)
- Data Manipulation Language statements (DML)
- Data Control Language statements (DCL)

DDL statements deal with creating, updating and deleting tables, indexes and relationships. Some examples are:

CREATE TABLE, ALTER TABLE, DROP TABLE

DML statements manipulate the data held in the tables. Some examples are:

SELECT, UPDATE, INSERT

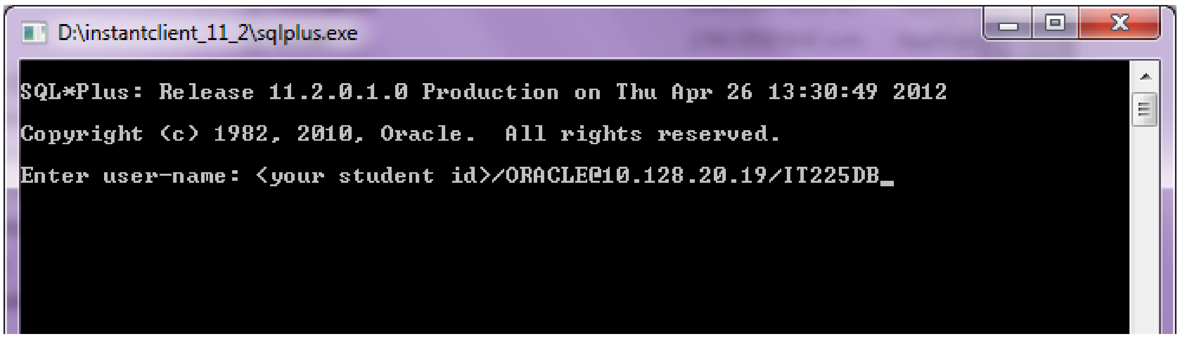
DCL statements include permission and transaction processing. Some examples are:

GRANT, REVOKE, COMMIT, ROLLBACK

Using ISQL*Plus

1. Copy the folder SQLPLUS from moodle to your Local Home drive.
2. You will find a folder by the name ***instantclient_11_2***. Open this folder and double click the SQLPLUS application.
3. Once the program has started and is showing the username prompt, type the following.
   (your username is your student id followed by the password and host).

   StudentId/ORACLE@10.128.20.19/IT225DB



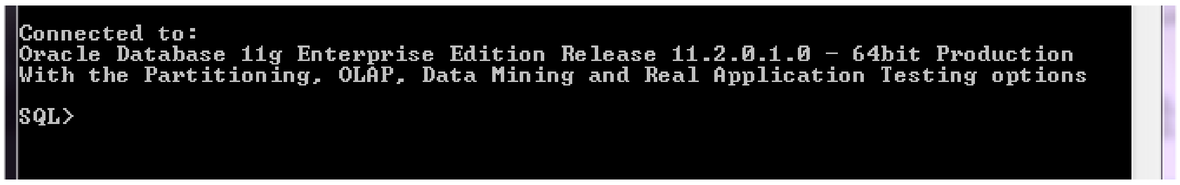
```
D:\instantclient_11_2\sqlplus.exe

SQL*Plus: Release 11.2.0.1.0 Production on Thu Apr 26 13:30:49 2012

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Enter user-name: <your student id>/ORACLE@10.128.20.19/IT225DB_
```

You should get a confirmation message as shown below indicating that you are connected to the Database.



```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

Tables

A relational database system contains one or more objects called ***tables***. The data or information for the database is stored in these tables. Each table should have a unique name and is made up of ***columns*** and ***rows***.

Columns are defined by the column name, data type, and any restrictions (called ***constraints***) it may have. Rows contain the records or data for the column.

Some simple rules to follow when you name the tables and columns:

- name must start with a letter and can be followed by letters, numbers, or underscores.
- name should not exceed a total of 30 characters.
- name should not use any SQL reserved keywords (such as "select", "create", "insert", etc.)

In this session we will create 5 tables called COLOUR, VEHICLE, CUSTOMER, SALESPERSON and SALESDETAIL.

Constraints

Constraints disallow some control on types of data in the rows of the table. There are several types of constraints.

<u>Primary Key</u>

Purpose: Ensures that value in a column cannot be null and has a unique occurrence.

So: Data in this field cannot be blank or cannot have a repeated value.

Snippet of code: `Cust_no    NUMBER(6)    PRIMARY KEY`

<u>Foreign Key</u>

Purpose: Ensures that value in a column belongs to another column in the connecting table or is null.

So: Data in this field should be present in the parent table or should be blank.

Snippet of code: `Colour    NUMBER(6)    REFERENCES colour(colour )`

<u>Not Null</u>

Purpose: Ensures that value in a column in not left blank.

So: Forces the field to have data in it.

Snippet of code: `F_name    VARCHAR(30)    NOT NULL`

<u>Value Constraint / Check Constraint</u>

Purpose: Ensures that all values in a column satisfy certain conditions.

So: Data in this field should comply with the value constraint.

Snippet of code: `Gender    CHAR(1)    CHECK (gender IN (‘M’, ‘F’))`

<u>Unique Constraint</u>

Purpose: Ensures that the data in this field has unique values or is null

So: Forces the column to have unique values only, or it could be blank.

Snippet of code: `Email    VARCHAR(30)    UNIQUE`

Default Constraint

Purpose:	Provides a default value for a column when none is specified
So:	Inserts the column to have a default value, which can be modified
Snippet of code:	`Rate NUMBER(3,1) DEFAULT 20.0`

Data Types

Some of the allowed data types are

Oracle	Use
VARCHAR	Variable length text. Recommended for most character fields. For example: model VARCHAR(10) will allow the *model* column to have a data type of VARCHAR. The number in the bracket gives the maximum length of the field. In this case, *model* has a field size of up to 10 characters (this could be numbers or alphabets or a combination of both)
CHAR	To store fixed length text and is more efficient than VARCHAR. For example: reg_no CHAR(6) will allow the *reg_no* column to store a fixed length of 6 characters (this could be numbers or alphabets or a combination of both)
NUMBER	Integers defined with the max number of digits, example can store up to $(-10^{38} + 1)$ to $(10^{38} - 1)$ A NUMBER field is used whenever some arithmetic is required on that field. For example: Year NUMBER(4) will allow the *Year* column to a 4 digit number (like 1234). The NUMBER field may sometimes consist of two numbers in the bracket. The first number still means the length of the entire field, and the second number indicates how many decimal places there are. An example of this is shown in the rate field Rate NUMBER(3,1) So for *rate*, the maximum value is 99.9 – three digits, with one of them being a decimal place.

DATE	Store a date format from 1-Jan-0001 to 31-Dec-9999 For example: date_sold DATE will allow for the *date_sold* column to store a date. Note that there is no bracket with values in it.
TIME	Store a time to the accuracy of 100 nano seconds
DATETIME	Stores date and time format from 1-Jan-1753 to 31-Dec-9999 including time with an accuracy of 3.33 milliseconds.

SQL Syntax

There are a few rules which **must** be followed, and a few which **may** be followed (recommended to aid clarity). This course requires these conventions to be followed.

- Every SQL command must be ended with a semicolon. There are some cases where a semicolon is not required but putting one in does not matter, so for the purpose of this course it is safer and less confusing to **use a semicolon at the end of all statements**.

- Data in fields with a data type of CHAR, VARCHAR or DATE need single quotes around it. Data in fields with a data type of NUMBER does not require anything around it.

- SQL **commands are not case sensitive**. That is, they can be typed in upper or lower case.
  On the other hand, **data within a table *is* case sensitive**. If a field contains the name ‘Katniss Everdeen’, you will not retrieve it by asking for ‘katniss everdeen’.

- Table names and field names must **not** contain spaces. In place of spaces, where two or more words are required, use the underscore character. For example:
  reg_no or RegNo

- SQL **commands are usually typed in upper case**, and the **table names or field names are usually typed in lower case**.
  Each new clause within a command is usually entered on a new line. These two conventions are not critical but they make a command much more readable.

Writing a script

It would be very useful to have a copy of all the SQL commands that you run. You may want to create the script on notepad and then run them in SQL-Plus.

To set your editor (to notepad) type in the following at the SQL prompt

```
SQL> DEFINE_EDITOR = notepad
```

Now you can write your script in a new file (or open a file if it already exists) by typing in the following at the SQL prompt

```
SQL> EDIT <yourname>_Lab1_Create.sql
```

If you do not type in the .sql in the end, the editor will add the extension for you. Use notepad to type in the commands to create the 5 tables in your database (these are given in pages 9-14).

To run the script, you may either

- **Run part of the script**

  Select the code in notepad, Ctrl+C to copy
  Move to the SQL window, Right click on the Title bar – Edit – Paste

- **Run the whole script**

  On the SQL prompt type
  ```
  SQL> START <yourname>_Lab1_Create.sql
  ```

Including comments in the script

You can insert comments using /* */

```
/*  DDL to create the vehicle table
    Code written by <your name>    */
```

Check the tables present in the database

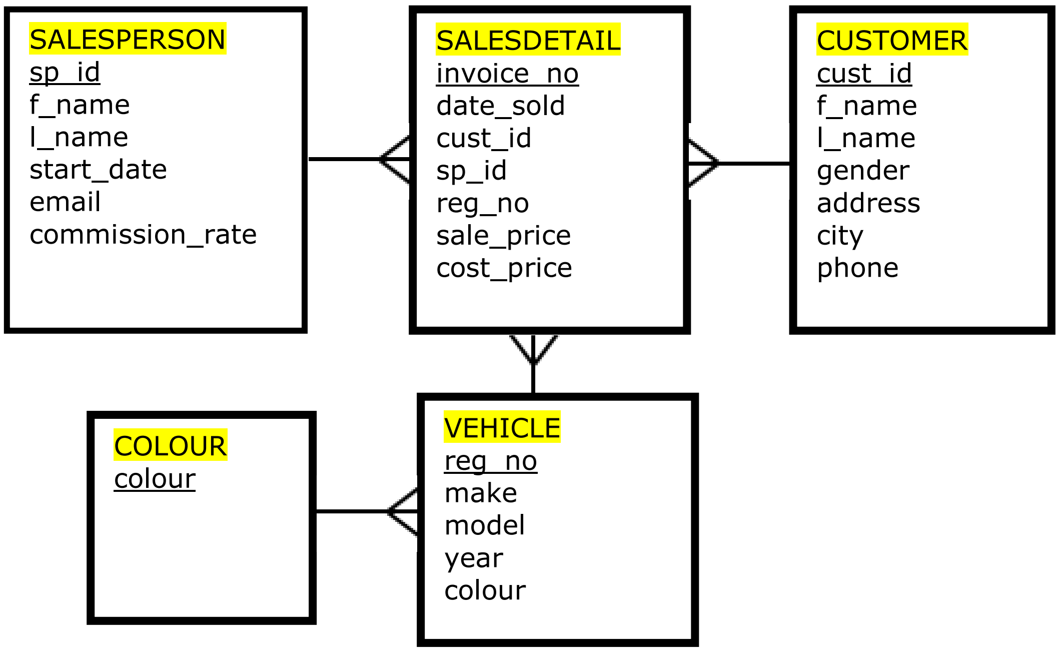
To check all the tables that have been created you can type, or type in notepad and paste from there.

```
SELECT table_name
FROM user_tables;
```

If you have any tables present, you may remove them. How do I remove a table?

```
DROP TABLE <table name>;
```

In this session we will create 5 tables called COLOUR, VEHICLE, SALESPERSON, CUSTOMER, and SALESDETAIL.



Creating a table

The syntax to create a table is as follows:
CREATE TABLE <table name>
(<column name> <data type> [<constraints for the column if any>]);

COLOUR

To create the COLOUR table type in

```
CREATE TABLE colour
(colour VARCHAR(10) PRIMARY KEY);
```

<u>Explanation of code:</u>

<u>Line 1</u>: CREATE TABLE <give the name of the table>

<u>Line 2</u>:
- Open bracket, remember to close this when all the attributes are defined
- Each attribute is defined by its **name**, followed by the **datatype**.

- Some datatypes are accompanied by the **size**. In this case, VARCHAR (10) means the colour attribute can hold up to a maximum of 10 characters
- Some attributes may have a **constraint** attached to it. The constraint is defined after the datatype. In this case, PRIMARY KEY
- Close bracket followed by semi colon to end the statement.

Running a script

To run a script type in the following at the SQL prompt

```
SQL> START <YOURNAME>_Lab1_Create.sql
```

To check all the tables that have been created you can type:

```
SELECT table_name
FROM user_tables;
```

To see the table structure type in:

```
DESCRIBE colour;
```

Help from system tables

user_tables is a table that is part of the system tables (meta data or data dictionary). Whenever a table is created, an entry is made in the user_tables table. To see the tables that have been created, type in

```
SELECT table_name
FROM user_tables;
```

user_constraints is another system table (meta data or data dictionary). Whenever a constraint is defined for an attribute, an entry is made in the user_constraints table. To see the constraint name and the condition it represents type in

```
SELECT constraint_name, search_condition
FROM user_constraints
WHERE table_name = 'COLOUR';
```

Note that the **table name is in upper case**, this is the way it is stored in the data dictionary.

Check out the details stored in these system tables by typing in

```
DESCRIBE USER_TABLES;
DESCRIBE USER_CONSTRAINTS;
```

VEHICLE

Open the script <yourname>_Lab1_Create.sql and add the code to create the VEHICLE table.

In the VEHICLE table, most of the field datatypes for this table are VARCHAR, which means they are variable length fields holding characters (which can be letters or numbers). Note that Reg_No is given a CHAR datatype as it holds data of a fixed length of 6 characters. Immediately after, in brackets, type the length of the field.

```
CREATE TABLE vehicle
(Reg_No      CHAR(6)          PRIMARY KEY,
 Make        VARCHAR(10)      NOT NULL,
 Model       VARCHAR(10)      NOT NULL,
 Year        NUMBER(4)        CHECK(year >= 2009),
 Colour      VARCHAR(10)      REFERENCES colour(colour));
```

Each column definition is separated by a comma, **no** comma for the last one.

Explanation of code:

Line 5: CHECK (<put in condition here>)

- The condition can use a mathematical operator
  Example: year > 2009
  year < 2009
  year = 2009
  year >= 2009
  year <= 2009

- The condition can use IN, OR, AND, BETWEEN expressions
  Examples: The following condition ensures that the vehicle has a year 2009, 2010, or 2011.
  year IN (2009, 2010, 2011)
  year = 2009 OR year = 2010 OR year =2011
  year >= 2009 AND year <= 2011
  year BETWEEN 2009 AND 2011

Line 6: Foreign key implementation

- Always starts with the word REFERENCES
- Give **table name** it refers to and **in brackets** give the **column name** it refers to.

SALESPERSON

The next table you have to create holds details of all the salespeople at the car yard. Details include the Id number of the salesperson, their name, the date they started working for the car yard, their email, and the commission rate he or she is paid when they sell a car. Note that the start_date of a salesperson is after 1-Jan-2005.

Type in as shown below:

```
CREATE TABLE salesperson
(Sp_Id          CHAR(2)             PRIMARY KEY,
 F_Name         VARCHAR(15)         NOT NULL,
 L_Name         VARCHAR(15)         NOT NULL,
 Start_Date     DATE                CHECK(start_date > '01-Jan-2005'),
 Email          VARCHAR(50)         UNIQUE,
 Rate           NUMBER(3,1)         DEFAULT 20.0 NOT NULL);
```

<u>Explanation of code:</u>

<u>Line 5</u>: The datatype for DATE will not have a size attached.
Note the format of the date used.
Dates have to be enclosed in single quotes

<u>Line 7</u>: NUMBER(3,1) indicates an NUMBER datatype with up to 3 digits, of which 1 is a decimal. This will hold values -99.9 to 99.9

Note that **2 constraints are defined** for the Rate attribute DEFAULT and NOT NULL. When more than one constraint is define, write the constraints together separated by a space.

CUSTOMER

The CUSTOMER table you have to create holds details of all the customers. Details include the Id number of the customer, their name, their gender, address, city and phone. Note that the **gender entered can only be M or F**. Remember SQL is case sensitive, so make sure you type in M or F in upper case.

```
CREATE TABLE customer
(Cust_Id        CHAR(3)             PRIMARY KEY,
 f_Name         VARCHAR(15)         NOT NULL,
 l_Name         VARCHAR(15)         NOT NULL,
 gender         CHAR(1)             CHECK(gender IN ('M', 'F')),
 Address        VARCHAR(15),
 City           VARCHAR(9),
 Phone          VARCHAR(8));
```

Explanation of code:

Line 5: In the CHECK condition, we have opened 2 brackets and make sure you close both the brackets.

Note that **SQL is case sensitive**, so make sure you type in M or F in upper case.

Foreign Key Constraints

The SALESDETAIL table has three foreign key constraints that enforce referential integrity between the tables. This will mean that we can only have a reg_no in the salesdetail table if the reg_no exists in the vehicle table. It also maintains this through updates and deletions. So that a vehicle cannot be deleted if it is referenced in the salesdetail table, also a reg_no cannot be modified if it is being used. These could have been changed using CASCADE DELETE or CASCADE DELETE referential integrity constraints.

SALESDETAIL

```
CREATE TABLE salesdetail
(Invoice_No   CHAR(4)           PRIMARY KEY,
Date_Sold     DATE,
Cust_Id       CHAR(3)           REFERENCES customer(cust_id),
Reg_No        CHAR(6)           REFERENCES vehicle(reg_no),
Sp_id         CHAR(2)           REFERENCES salesperson(sp_id),
Sale_Price    NUMBER(7,2),
Cost_Price    NUMBER(7,2));
```

Note that the tables must be created in the right sequence; attempts to reference a table before it has been created will result in an error.

Running a script

To run a script type in the following at the SQL prompt

```
SQL> START <YOURNAME>_Lab1_Create.sql
```

To check all the tables that have been created you can type:

```
SELECT table_name
FROM user_tables;
```

LAB SESSION 2: POPULATING TABLES

Loading data into a table

The syntax to insert data into a table in the order the columns were created is as follows:

```
INSERT INTO <table name> VALUES (‘value 1’, ....)
```

The syntax to insert data into a table in the order you wish to enter values is as follows:

```
INSERT INTO <table name> (column names) VALUES (‘value 1’, ....)
```

Write a new script to store the commands to insert values by typing in the following

```
SQL> EDIT <yourname>_Lab2_Insert.sql
```

Colour table

Before we start to enter any vehicles in we need to populate the look up table for the colours.

Colour
Red
White
Green
Blue
Silver
Black
Purple

To load this data in we can type

```
INSERT INTO colour VALUES ('Red');
```

Note:

- fields defined as NUMBER do not require single quotes, but DATE, CHAR and VARCHAR fields do.
- the values entered are case sensitive (if you type in 'red' instead of 'Red', it will be considered to be a new colour).

Write the SQL statements to insert the remaining rows in the COLOUR table.

To save these changes we need to type

```
COMMIT;
```

To see the values in the table, type in

```
SELECT *
FROM colour;
```

Test Plan:

It is always necessary to test the data that will be stored in the database. We shall check if the constraints that we placed on the columns are working. A sample test plan can be as follows:

Constraint	Condition	Test case	Expected result	Actual result
Primary key (Colour)	Should not be null	Enter a colour with no value	Should not accept	Did not accept
	Should not allow duplicates	Repeat a colour – enter Red again	Should not accept	Did not accept

Vehicle table:

Before we start to enter any SALESDETAIL in we need to populate the VEHICLE table.

Reg_no	Make	Model	Year	Colour
GKN534	Mazda	626	2014	Red
ALP394	Nissan	Bluebird	2012	White
NT6776	Toyota	Corolla	2011	Green
GLM123	Honda	Accord	2010	Blue
OM1122	Mazda	323	2012	Purple
RS3456	Mazda	323	2013	White
ZHU123	Nissan	Note	2009	White
PRH345	Honda	Accord	2013	Red
SUT143	Nissan	Bluebird	2013	Silver
SALES1	BMW	525	2014	Black

To load this data in we can type

```
INSERT INTO vehicle VALUES ('RS2533', 'Mazda', '626', 2011, 'Red');
```

Note that all VARCHAR data has to be set within single quotes.

One of the problems with this syntax is that we need to specify all the fields and have them in the right order. If the table were to change, the insert statement would not work, or could put the data in incorrectly, for example a change in the order of make and model could be problematic. We should always state the order of the fields as in

```
INSERT INTO vehicle (reg_no, make, model, year, colour)
VALUES ('PK8829', 'Toyota', 'Corolla', 2014, 'White');
```

Write the SQL statements to insert the remaining rows in the VEHICLE table. To see the values in the table, type in

```
SELECT *
FROM vehicle;
```

Test Plan for Vehicle table:

Constraint	Condition	Test case	Expected result	Actual result
Primary key (Reg_no)	Should not be null	Enter a reg_no with no value	Should not accept	Did not accept
	Should not allow duplicates	Repeat a reg_no – try to enter GKN534 again	Should not accept	Did not accept
Not Null (Make)	Should not be null	Enter a make with no value	Should not accept	Did not accept
Not Null (Model)	Should not be null	Enter a model with no value	Should not accept	Did not accept
Check (Year)	Should not be before 2009	Year >2009 eg. 2010	Should accept	Did accept
		Year = 2009	Should accept	Did accept
		Year < 2009 Eg. 2008	Should not accept	Did not accept
Foreign key (Colour)	Should not be null	Enter a colour with no value	Should not accept	Did not accept
	Should not allow duplicates	Repeat a colour – try to enter Red again	Should not accept	Did not accept

Salesperson table:

Sp_Id	f_Name	l_name	Start_Date	Email	Rate
1	Michael	Knapp	10-Jan-12	michael.knapp@wcd.co.nz	12.5
2	Bradley	Palmer	24-Mar-12	bradley.palmer@wcd.co.nz	10
3	Kane	Hunter	12-May-13	kane.hunter@wcd.co.nz	12.5
4	<put in your details ☺>				10

Note that fields defined as NUMBER do not require single quotes, but CHAR, DATE and VARCHAR fields do. When you have entered all of the data for the vehicles table, and if you enjoy typing do the same for the other three tables. However, for your own sanity it will be best to populate your schema with the same tables as everyone else.

```
INSERT INTO salesperson
VALUES (1, 'Michael', 'Knapp', '10-Jan-2012', 'michael.knapp@wcd.co.nz', 12.5);
```

Write the SQL statements to insert the remaining rows in the SALESPERSON table. To see the values in the table, type in

```
SELECT *
FROM salesperson;
```

Test Plan for Salesperson table

Constraint	Condition	Test case	Expected result	Actual result <check & fill in>
Primary key (sp_id)	Should not be null	Enter a sp_id with no value	Should not accept	
	Should not allow duplicates	Repeat a sp_id – try to enter 1 again	Should not accept	
Not Null (f_name)	Should not be null	Enter f_name with no value	Should not accept	
Not Null (l_name)	Should not be null	Enter l-name with no value	Should not accept	
Check (start_date)	Should be after 1-Jan-2005	Start_date > 1-Jan-2005 eg. 12-Jan-2005	Should accept	
		Start_date = 1-Jan-2005	Should not accept	
		Start_date < 1-Jan-2005 eg. 1-Jan-2004	Should not accept	
Unique (Email)	Should not allow duplicates	Repeat a colour – try to enter existing email again	Should not accept	
Default (rate)	Should default to 20.0	Insert a row with entries for other attributes of table.	Should insert default rate	

Customer table

Cust_Id	f_Name	l_Name	Gender	Address	City	Phone
101	Katniss	Everdeen	F	45 Dinsdale Rd	Hamilton	8123456
102	Peeta	Mallark	M	123 Anglesea St	Hamilton	8111111
103	Gale	Hawthorne	M	717 River Rd	Hamilton	8221122
104	Haymitch	Abernathy	M	24 Duke St	Cambridge	8881888
105	Primrose	Everdeen	F	RD1	Cambridge	8112211
106	Effie	Trinket	F	655 Tristram St	Hamilton	8181818
107	<put in your details ☺>					

Write the SQL statements to insert the rows in the CUSTOMER table. When you are finished, to see the values in the table, type in

```
SELECT *
FROM customer;
```

Test Plan for Customer table

Constraint	Condition	Test case	Expected result	Actual result
Primary key (cust_id)	Should not be null			
	Should not allow duplicates			
Not Null (f_name)	Should not be null			
Not Null (l_name)	Should not be null			
Check (gender)	Should be after 1-Jan-2005			

Salesdetail table

Invoice_No	Date_Sold	Cust_Id	Reg_No	Sale_Price	Cost_Price
3144	3-Apr-14	101	ALP394	14500	8750
3145	3-May-14	102	NT7667	16000	14840
3146	7-Jun-14	103	RS3456	23000	19000
3147	12-Jun-14	104	ZHU123	17995	15875
3148	14-Jul-14	105	SALES1	29200	16900
3149	17-Jul-14	102	GKN533	21000	18675

Write the SQL statements to insert the rows in the SALESDETAIL table.

Test Plan for Salesdetail table

Constraint	Condition	Test case	Expected result	Actual result
Primary key (invoice_no)	Should not be null			
	Should not allow duplicates			
Foreign key (cust_id)	can be null			
	Should exist in parent table	Value in parent table		
		Value not in parent table		
Foreign key (reg_no)	can be null			
	Should exist in parent table	Value in parent table		
		Value not in parent table		
Foreign key (sp_id)	can be null			
	Should exist in parent table	Value in parent table		
		Value not in parent table		

Updating Rows

Before we change the contents of a table, we will make sure that we have saved all the data. All changes to data need either a COMMIT if the change is successful or a ROLLBACK if unsuccessful. A COMMIT will save commit the changes so all users can see the changes while a ROLLBACK will undo this and all the other changes since the last COMMIT.

To change all the records in a table we use the update command

```
COMMIT;
UPDATE salesperson
SET rate = 16;
```

Now type

```
SELECT *
FROM salesperson;
```

What is the rate?
Now type

```
ROLLBACK;
SELECT *
FROM salesperson;
```

What is the rate?

Updating Selected Rows

To change the value in a particular field for only a selection of rows: This is useful when needing to modify a large set of records where they all share a particular criteria

```
UPDATE vehicle
SET make = 'BMW'
WHERE make = 'bmw';
```

Be careful with update queries and use a select query first to check whether you are updating the right ones. If the WHERE clause had been omitted then ALL the cars would have a make of BMW. Always check the data after a change and COMMIT or ROLLBACK as necessary.

To change the value in a particular field for a single row then the WHERE clause must have a candidate key.

```
UPDATE vehicle
SET colour = 'Red',   Year = 2010
WHERE Reg_no = 'ZHU123';
```

Deleting all records

Records are usually archived to a data warehouse and only transactional records deleted. The following command will empty a table of all records: for example, this would not work for the cars that were sold as the SALESDETAIL table references these.

```
DELETE FROM vehicle;
```

Deleting selected records

```
DELETE FROM vehicle
WHERE year < 2009;
```

****

EXERCISE 2

Modify the script <yourname>_Lab2_Insert.sql to include the following tasks. Run it using SQLPLUS and check the results. When you have completed all the tasks in the exercise, show it to your tutor

1. Save the changes you have made. Update Michael Knapp's commission rate to 18%.
2. Update all the sales people so they have a new commission rate of 19%. Undo this update.
3. Delete all the customers, then decide it was not a good idea and to rollback.

****

Revision

1. When entering in some values to test the data you get this error message

   ORA-00001: unique constraint (SYSTEM.SYS_C0063690) violated

   Explain what the problem would be?

2. On inserting other records you get this error

   ORA-02290: check constraint (SYSTEM. SYS_C0063691) violated

   Explain what the problem would be?

3. On inserting other records you get this error

   ORA-02291: integrity constraint (SYSTEM. SYS_C0063693) violated - parent key not found

   Explain what the problem would be?

4. On inserting other records you get this error

   ORA-02290: null constraint (SYSTEM. SYS_C0063692) violated

   Explain what the problem would be?

5 Match the following

	SQL command
a	CREATE
b	INSERT
c	SELECT
d	DESCRIBE
e	DELETE
f	DROP
g	UPDATE
h	ALTER
i	COMMIT
j	SAVE
k	ROLLBACK

	Used to do this task
1	Save changes made so far
2	Remove the contents of a table
3	View the contents of a table
4	Make a new table
5	Make a new row
6	View the structure of a table
7	Remove the entire table
8	Replace the value in a table
9	Modify the structure of a table
10	Undo the changes made
11	Not a valid command

6. Kane tries to delete a list of 3 order lines by copying this script into the command prompt.

```
DELETE FROM order_lines;
WHERE order_id = 1012;
```

   What he sees is this

```
SQL> delete from order_lines;

2413489 rows deleted.

SQL> where order_id = 1012;
SP2-0734: unknown command beginning "where orde..." - rest of line
ignored.
SQL>
```

   What has he done wrong? How can he correct this situation?

LAB SESSION 3: RETRIEVING DATA FROM A TABLE

In this section we will build up the vocabulary we can use to do the manipulation.

To display results, use the SELECT command. The most common syntax is

```
SELECT list of fields     (PROJECTION restricts the columns)
FROM table names
WHERE                     (SELECTION restricts the rows)
ORDER BY fields           (SORTS the rows)
```

Write a new script to store the commands to retrieve data by typing in the following

```
SQL> EDIT <yourname>_Lab3_Select.sql
```

The command below specifies that all fields from the vehicle table are to be listed, by using an asterisk. This command also has a FROM clause which specifies the table from which the data is to be selected.

```
SELECT *
FROM vehicle;
```

Note that this could be typed on just one line but setting it out like this is part of the SQL convention discussed on page 4. You should see the following rows.

Reg_no	Make	Model	Year	Colour
GKN534	Mazda	626	2014	Red
ALP394	Nissan	Bluebird	2012	White
NT6776	Toyota	Corolla	2011	Green
GLM123	Honda	Accord	2010	Blue
OM1122	Mazda	323	2012	Purple
RS3456	Mazda	323	2013	White
ZHU123	Nissan	Note	2010	Red
PRH345	Honda	Accord	2013	Red
SUT143	Nissan	Bluebird	2013	Silver
SALES1	BMW	525	2014	Black

Projection

SQL is a complete relational algebra. This means we can manipulate "relations" to get new relations. To display only certain fields, list them all after SELECT. Put a comma between each field, but not one after the last field name:

```
SELECT Make, Colour
FROM vehicle;
```

Operators

To display only certain records, add a WHERE clause. This specifies the criterion or criteria that are to be used in selecting records. Simple conditions typically contain one of the SQL comparison operator listed below:

operator	description	SQL example	Expected result

=	Equal to	SELECT *
		FROM vehicle
		WHERE year = 2013;
			3 rows

>	Greater than	SELECT *
		FROM vehicle
		WHERE year > 2013;
			2 rows

>=	Greater than or equal to	SELECT *
		FROM vehicle
		WHERE year >= 2013;
			5 rows

<	Less than	SELECT *
		FROM vehicle
		WHERE year < 2013;
			5 rows

<=	Less than or equal to	SELECT *
		FROM vehicle
		WHERE year <= 2013;
			8 rows

< >	Not equal to
	
	Same as !=
		SELECT *
		FROM vehicle
		WHERE year <> 2013;
			7 rows

BETWEEN	Between the values specified, inclusive of the values	SELECT *
		FROM vehicle
		WHERE year BETWEEN
		2010 AND 2012;
			5 rows

Using AND, OR, NOT clause

The following statement will list only one record. Not only are we looking just for white cars, now they must have a value of “Mazda” in the Make field.

```
SELECT reg_no, make, model
FROM vehicle
WHERE colour = ‘White’
AND make = ‘Mazda’;
```

The following statement lists white vehicles and red vehicles in the yard.

```
SELECT reg_no, make, model
FROM vehicle
WHERE colour = ‘White’
AND colour = ‘Red’;
```

Why are there no rows selected? AND is used to satisfy both conditions. There is no vehicle that is both White and Red and so no rows are selected.

If you wish to show all white vehicles along with the red vehicles use the following statement which should return 6 rows.

```
SELECT reg_no, make, model
FROM vehicle
WHERE colour = ‘White’
OR colour = ‘Red’;
```

A NOT operator can be used to negate the condition.

```
SELECT reg_no, make, model
FROM vehicle
WHERE NOT colour = ‘White’;
```

To display records where the year is between 2010 and 2012 (inclusive of 2010 and 2012), can also be achieved by the following

```
SELECT *
FROM vehicle
WHERE year >= 2010 AND year <=2012;
```

****

EXERCISE 3

Modify the script <yourname>_Lab3_Select.sql to include the following tasks. Run it using SQLPLUS and check the results. When you have completed all the tasks in the exercise, show it to your tutor

1. Show all details of the purple vehicles in the yard.
2. Show all details for male Hamilton customers.
3. Show all details for female customers who live in Hamilton.
4. Show all details for female customers or those that live in Hamilton.
5. Show all details for female customers and those that do not live in Hamilton.
6. Show all details for all vehicles with a Year of 2008 or 2009.
7. Show reg. number and make for all vehicles that are made between 2010 and 2012.
8. Show invoice number, date sold and registration number for all sales over $16,000.

9. Show all the vehicles that have a make of either Mazda or BMW.
10. Show all details of the sale for vehicles priced over $14,500.
11. Show invoice number, date sold and registration number for all sales over $14,500 but below $18,000.
12. Show all details of sales in June 2014.
13. Show all details of sales that were not in May 2014.
14. Show all salespeople started in 2012. (Use a BETWEEN clause)

****

Column Headings

SQL uses the column names to display query results. Since some of the column names are short or cryptic, it could be hard to understand. You can give it a useful column heading using the HEADING clause with the COLUMN command.

```
COLUMN column name HEADING column heading
```

To get an output from VEHICLE with new heading for Reg_no, type in the following command

```
COLUMN Reg_no HEADING Registration
SELECT Reg_no, Make
FROM vehicle;
```

The following heading will be displayed

Registration	Make

Note: This headings will remain in effect until you enter another heading for the columns, reset each column's format, or exit from SQL.

To change a column heading to two or more words, enclose the new heading in single or double quotation marks when you enter the COLUMN command. To give the column Reg_no the heading *Registration Number*

```
COLUMN Reg_no HEADING "Registration Num"
SELECT Reg_no, Make
FROM vehicle;
```

The following heading will be displayed

Registration Num	Make

To display a column heading on more than one line, use a vertical bar (|) where you want to begin a new line. To give the column Reg_no the heading *Registration Number* in two lines, type in

```
COLUMN Reg_no HEADING Registration | Number
SELECT Reg_no, Make
FROM vehicle;
```

The following heading will be displayed

Registration Number	Make

Formatting NUMBER Columns

When displaying NUMBER columns, you can either accept the SQL*Plus default display width or you can change it using the COLUMN command. A NUMBER column's width equals the width of the heading or the width of the FORMAT plus one space for the sign, whichever is greater.
`
The COLUMN command identifies the column you want to format and the model you want to use, as shown below:

```
COLUMN column name FORMAT model
```

Use *format model* to add commas, dollar signs, and/or leading zeros to numbers in a given column. A *format model* is a representation of the way you want the numbers in the column to appear, using 9's to represent digits.

To display cost_price with a dollar sign, a comma, and the numeral zero instead of a blank for any zero values, enter the following command:

```
COLUMN cost_price FORMAT $99,999
SELECT Invoice_no, reg_no, cost_price
FROM salesdetail;
```

Now run the query.

Note:
The *format model* will stay in effect until you enter a new one, reset the column's format with

```
COLUMN column name CLEAR
```

or exit from SQL*Plus.

LAB SESSION 4: SELECT STATEMENTS

Using LIKE clause

LIKE command is used to search for a pattern. A wild card is a special character used when you are trying to select records with the LIKE command. For instance:

```
SELECT *
FROM customer
WHERE phone LIKE ‘81%’;
```

This will select all customers with a phone number beginning with the numbers 81. You can see, therefore, that the % (percent) sign means *“I don’t care what or how many characters follow the ones that I have specified”* (in this case, 81).

Have a look at another example, this time selecting from the VEHICLE table:

```
SELECT *
FROM vehicle
WHERE colour LIKE ‘B%’;
```

This will select 2 records: GLM123, because it is Blue, and SALES1, because it is Black. It has selected anything whose colour starts with the letter B, no matter how many letters follow the B. It is important to note that even if there were no letters following the B, the record would be selected. *The % sign therefore stands for “zero, one or many” characters.*

A second wild card is the _ (underscore) character. This means “any character of any value, but only one character”. This will only select 1 record, because it is looking for any vehicle with a colour starting with B, and exactly 4 letters long - blue is selected, black is not.

```
SELECT *
FROM vehicle
WHERE colour LIKE ‘B_ _ _’;
```

Wild cards can be used together. For instance:

```
SELECT *
FROM customer
WHERE f_name LIKE ‘_a%’;
```

This will select all customers whose name has a second letter of “a” and a varying number of letters following the “a”. Records selected will be for K<u>a</u>tniss, G<u>a</u>le and H<u>a</u>ymitch. However, if we had specified

```
SELECT *
FROM customer
WHERE f_name LIKE '%a%';
```

It would have selected those three people plus Peet<u>a</u>, because it is now selecting people with any number of letters before the "a", not just one letter before the "a".

As well as using LIKE, you can select using the opposite NOT LIKE:

```
SELECT *
FROM customer
WHERE f_name NOT LIKE 'P%';
```

This means: select all customers whose names do not start with P.

Aggregate functions

Aggregate functions enable you to add up columns, average columns, count occurrences, select a maximum value or a minimum value. The keywords are SUM, AVG, COUNT, MAX, MIN.

Try out the following commands and observe how they work:

```
SELECT SUM(sale_price)
FROM salesdetail;

SELECT SUM(sale_price)
FROM salesdetail
WHERE sp_id = 2;

SELECT AVG(sale_price)
FROM salesdetail;

SELECT AVG(sale_price)
FROM salesdetail
WHERE sp_id = 2;

SELECT AVG(sale_price), AVG(cost_price)
FROM salesdetail;

SELECT COUNT(invoice_no)
FROM salesdetail;

SELECT COUNT(*)
FROM salesdetail;

SELECT MAX(sale_price), MIN(sale_price)
FROM salesdetail;
```

They cannot be mixed with fields or non-aggregate functions (without the GROUP BY clause).

Column Alias

Some group functions can be used with arithmetic expressions. The following example will give the total of the difference between sale price and cost price. This will produce just one line of output.

```
SELECT sale_price - cost_price
FROM salesdetail;
```

You can still use a column alias if you wish:

```
SELECT sale_price - cost_price "Total Profit"
FROM salesdetail;
```

****

EXERCISE 4

Modify the script <yourname>_Lab4_Aggregate.sql to include the following tasks. Run it using SQLPLUS and check the results. When you have completed all the tasks in the exercise, show it to your tutor

1. List the sales details and the make for all sales.
2. Give a count of all customers.
3. Give a count of all customers who live in Hamilton.
4. Give the average rate.
5. List the total sale price and the total cost price of all invoices in the sales details.
6. List the total sale price and the total cost price of all invoices in the sales details for all sales in 2010.
7. The sale price is increased by 10%. Show the current sale price and the new sale price.
8. The sale price is discounted by 5%. Show the current sale price and the discounted price.
9. Show the first name, last name, current rate and new rate (add 0.5 to current rate) of all salespeople that started in 2010.
10. Show how old each vehicle is by using a calculation on the age column (to keep it simple, you may type in the current year).

****

What did we learn?

Use the following table STUDENT to determine the output of the following statements.

stu_Id	stu_name	prog_code	gender	dob
1	Tris Prior	Database	F	06-Feb-96
2	Four Eaton	Programming	M	16-Mar-92
3	Caleb Prior	Database	M	22-May-95
4	Tori Wu	Multimedia	F	12-Jun-80
5	Peter Hayes	Programming	M	08-Aug-95

```
a) .. WHERE dob >= ’22-May-95’;

b) ... WHERE dob > ’22-May-95’;

c) ... WHERE dob < ’22-May-95’;

d) ... WHERE dob <= ’12-Jun-80’;

e) ... WHERE stu_name LIKE ‘P%’;

f) ... WHERE stu_name LIKE ‘%T%’;

g) ... WHERE stu_name LIKE ‘% P%’;

h) ... WHERE stu_name LIKE ‘%s’;

i) ... WHERE stu_name LIKE ‘%s%’;

j) ... WHERE stu_name LIKE ‘%s %’;

k) ... WHERE stu_name LIKE ‘_a%’;

l) ... WHERE stu_name LIKE ‘%o_’;

m) ... WHERE dob BETWEEN ’01-Feb-95’ AND ’31-Dec-96’;

n) ... WHERE dob > ’01-Feb-95’;

o) ... WHERE dob IN (’01-Feb-95’,’31-Dec-96’);
```

LAB SESSION 5: JOINS

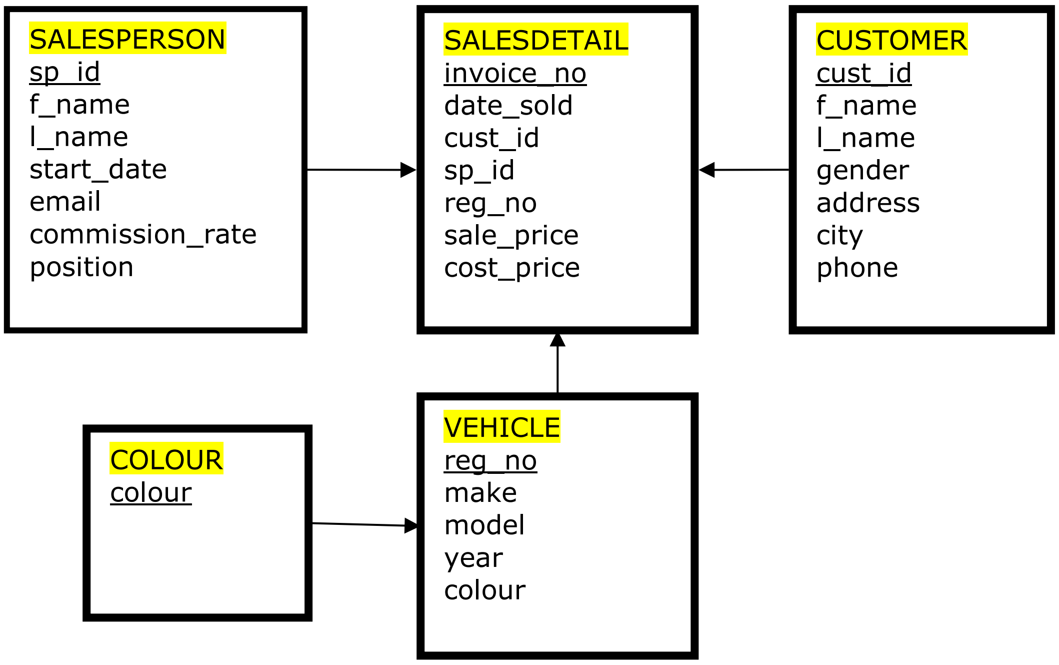
Create a new script to store the commands to retrieve data using joins and nested queries

```
SQL> EDIT <yourname>_Lab5_Joins.sql
```

Selection from Multiple Tables

Where we have a one-to-many relationship it is often important to get the data from more than one table. If the process of normalisation is done with no loss of information, it is always possible to produce select queries that we need for a report.

For example, if we want to get details on the sales and the names of the salespeople we need to use two tables. As a first attempt we can get the sales information with just the salespersons id from the SALESDETAIL table. Always have the tables and their relationships in front of you.



Using Inner Joins (Joins)

The inner join selects all rows from both tables as long as there is a match between the columns in both tables

If we now try a select query from more than one table you get a product of all the different ways to combine the rows of one table with the other.

```
SELECT invoice_no, reg_no, f_name
FROM salesdetail, salesperson;
```

This is a ***cartesian product*** and we get every combination of sale and sales person. However, we want to restrict the results to the *Natural Join*. That is the name of the sales person that sold the car.

```
SELECT invoice_no, reg_no, f_name
FROM salesdetail, salesperson
WHERE sp_id = sp_id;
```

This results in an ***ambiguity error*** as we need to define what table the sp_id comes from. Note that the field that we are matching both tables on has the same name in each table – sp_id. SQL cannot cope with duplicate field names in the same command, so to make them unique, the table name is prefixed onto each field name and joined with a dot.

```
SELECT invoice_no, reg_no, f_name
FROM salesdetail, salesperson
WHERE salesdetail.sp_id = salesperson.sp_id;
```

There is a way to cut down on some of the typing – you can give each table name an abbreviation in the FROM clause and use it elsewhere in that query. In the following example, the SALESDETAIL table has been abbreviated to sd and the SALESPERSON table has been abbreviated to sp.

```
SELECT invoice_no, reg_no, f_name
FROM salesdetail sd, salesperson sp
WHERE sd.sp_id = sp.sp_id;
```

A form rule to use is that each foreign key is equal to the primary key in the table it references. Having an ERD with the key names makes writing SQL queries from more than one table extremely easy. So to get a list of the sales with the customer's names we need to be aware that the cust_id in the sales_details table is referencing the cust_id in the customers table.

```
SELECT sd.invoice_no, c.f_name
FROM salesdetail sd, customer c
WHERE sd.cust_id = c.cust_id;
```

Note that the abbreviation sd and c can be used on the first line even though it is not defined until the second line.

Using VIEW

A view is a SQL statement that is stored in a database with an associated name. View is also known as a virtual table.

- A view may contain all or selected rows of a table
- A view can be created from one or many tables

Views allow the user to

- Structure data in a way so it is user friendly
- Restrict access to data such that user can see and (sometimes) modify exactly what they need and no more.
- Summarize

Sometimes it is easier to solve a problem in two steps. The CREATE VIEW command will create a virtual table (a "view") which can then be used as input into a second query. The table is virtual in that it can be used like a table but does not store any data itself. It is not a temporary table, those are created differently.

Some reasons for creating a view:

- You may have a number of queries to do which involve looking just at Mazda's, so you create a subset of the vehicles table first then do all the queries off that one;
- You may be giving someone else access to only a subset of a table because you do not want them to see confidential material. For instance, you may create an employee_details table from an employee's table where the employee_details table holds no salary information.

For instance, we may want to know the sales history with the car details. The sales_details table tells us the registration numbers of the cars that have been sold but we don't know the make unless we join to the vehicles table. So we could say:

This aids the development effort as the view might be used in many reports and reports in an

```
CREATE VIEW vehicle_salesdetail AS
SELECT invoice_no, sd.reg_no, make, sale_price
FROM salesdetail sd, vehicle v
WHERE sd.reg_no = v.reg_no;
```

Alternatively we may want to hide the sales people table as it contains details like the commission rate but make public the contact details.

```
CREATE VIEW sales_people_details AS
SELECT sp_id, f_name, l_name, email
FROM salesperson;
```

****

EXERCISE 5

Modify the script <yourname>_Lab5_Joins.sql to include the following tasks. Run it using SQLPLUS and check the results. When you have completed all the tasks in the exercise, show it to your tutor

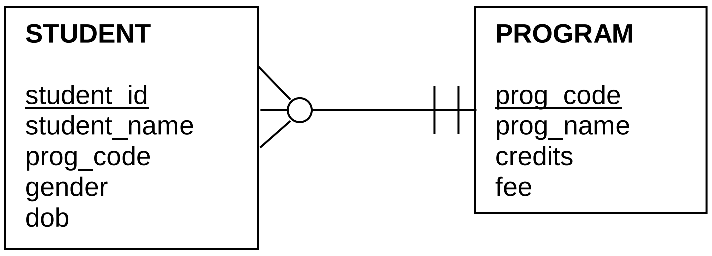
1. List all invoices with the invoice no, the date sold, the salesperson's first name and last name.
2. List all invoices with the invoice no, the date sold, the customer's first name and last name.
3. List all invoices with the invoice no, the date sold, the customer's first name and last name where the customer is from Cambridge.

****

What did we learn?

1. What is a Cartesian product?

2. Given the following model, answer Question a) to f)



a) Identify the problem with the following statement

```
SELECT student_id, prog_code, prog_name
FROM student s, program p
```

b) What is the purpose of having the **p** after **program** in line 2?

c) Identify the problem with the following statement

```
SELECT student_id, p.prog_code, prog_name
FROM student s, program p
```

d) What is the purpose of having the **p.** in front of the prog_code in line 1?

e) Why is it not necessary to use **p.** in front of the prog_name in line 1?

f) Identify the problem with the following statement

```
SELECT student_id, p.prog_code, prog_name
FROM student s, program p
WHERE p.prog_code = s.prog_code;
```