

Topics

1. PL/SQL review
2. Cursors
3. Functions
4. MongoDB

PL/SQL Review

- What are the different types of iteration statements?
- Basic LOOP
- FOR LOOP
- Cursor FOR LOOP
- WHILE LOOP
- How can you leave a loop
- EXIT
- EXIT WHEN
- What does Continue or Continue When do?

Select Into

- The Select Into statement used in a procedure's executable block can have three outcomes
- It retrieves a row
- No Data Found
- Too Many Rows
- If more than one row can be returned
- Use a Cursor

CURSORS

- Cursors are used to process multiple rows in PL/SQL blocks.
- In this course, we learn basic fundamentals about cursors. We use cursors to return multiple rows from a PL/SQL procedure to a caller procedure or program.
- A cursor is a pointer to a context area that includes the result of a processed SQL statement.
- Simply, a cursor contains the rows of a select statement.
- In PL/SQL, cursors are used to access and process the rows returned by a SELECT statement.
- There are two types of cursors:
 - Implicit cursors
 - Explicit cursors

Implicit Cursors

- The implicit cursors are automatically created while a statement such as INSERT, UPDATE, DELETE, and SELECT are executed.
- The implicit cursor attributes can be used to determine if any rows have been affected as a result of the execution of a SQL statement.
- In PL/SQL, the SELECT INTO statement, the implicit cursor can be evaluated to see if any row is returned.

Implicit Cursors

Attributes	Description
SQL%FOUND	It returns true if at least one row is affected by the execution of a DML or a SELECT statement
SQL%NOTFOUND	It returns true if no row is affected by the execution of a DML or a SELECT statement
SQL%ROWCOUNT	It returns the number of rows affected by the execution of a DML statement or a SELECT statement
SQL%ISOPEN	SQL%ISOPEN always has the value FALSE.

- Exceptions with SELECT INTO
- NO DATA FOUND
- TOO MANY ROWS RETURNED
- If multiple rows are returned we replace SELECT INTO with a SELECT within a CURSOR
- Explicit Cursor

Explicit Cursors

- The explicit cursors are defined in the declaration section of a PL/SQL block by programmers. It is used to process the multi-row results from a SELECT statement.
- To define cursor:
 - **CURSOR** cursor_name **IS** select_statement;
- To use an explicit cursor in a PL/SQL block:
 - Declare the cursor
 - The memory is allocated for the cursor
 - Open the cursor
 - Retrieve data from the cursor
 - Close the cursor
 - The memory allocated to the cursor is released.

Explicit Cursors - DECLARE

- Cursors can be defined in the declaration section.
- **CURSOR** cursor_name **IS** select_statement;
- DECLARE
- CURSOR cursor_1 IS
- SELECT last_name, job_title FROM employees
- WHERE job_title LIKE 'VP%'
- ORDER BY last_name;
- After the cursor is opened and a temporary table is loaded, you want to fetch the row column data a row at a time and put it into your appropriate variables.
- You can use **%type** to pick up the same definition for your variable that is used in the original table column

Cursor Defined

- You specify local variables and define the cursor in the declaration section.
- DECLARE
- e_last_name employees.lastname%type;
- e_job_title employees.jobtitle%type;
- CURSOR emp_cursor IS
- SELECT lastname, jobtitle
- FROM employees
- WHERE jobtitle LIKE 'VP%'
- ORDER BY lastname;

Open a Cursor

- You open and use the cursor in the executable section.
- **BEGIN**
- **OPEN emp_cursor;**

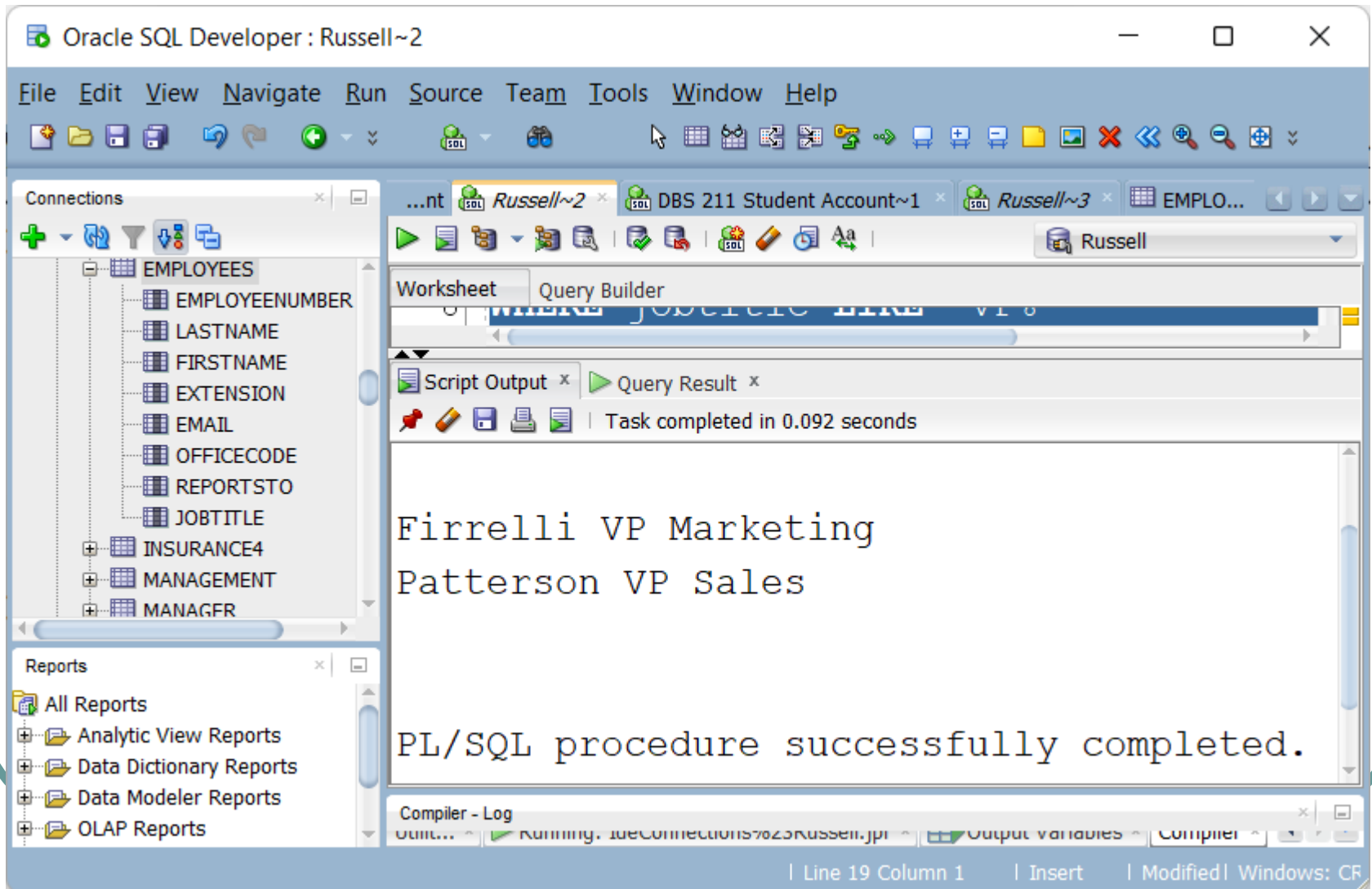
Fetching Data using a Cursor

- In the executable section, you can fetch data from a cursor row by row in a loop.
- The syntax is:
- `FETCH cursor_name INTO variable_list;`
- `BEGIN`
- `OPEN emp_cursor;`
- `LOOP`
- `FETCH emp_cursor into e_last_name, e_job_title;`
- `EXIT WHEN emp_cursor%notfound;`
- `dbms_output.put_line(e_last_name || ' ' || e_job_title);`
- `END LOOP;`

Close a Cursor

- You close a cursor in the executable section.
- BEGIN
- OPEN emp_cursor;
- LOOP
- FETCH emp_cursor into e_last_name, e_job_title;
- EXIT WHEN emp_cursor%notfound;
- dbms_output.put_line(e_last_name || ' ' || e_job_title);
- END LOOP;
- CLOSE emp_cursor;
- END;

Executing the Cursor Code



Explicit Cursor with Parameters

- You can pass parameters that affect what rows are returned when the cursor is opened.
- If you want to return all products with prices between two prices, you can pass an upper and lower price
- The following anonymous block has hard coded 115 and 120 as the upper and lower price to be used by the cursor

%Type

- When you use **%type**, you do not have to know the definition of a column in a table when setting up your variable.
- Also, if the type definition for that column changes, your code does not have to be updated – it will automatically pick up the new type definition

Explicit Cursor with Parameters

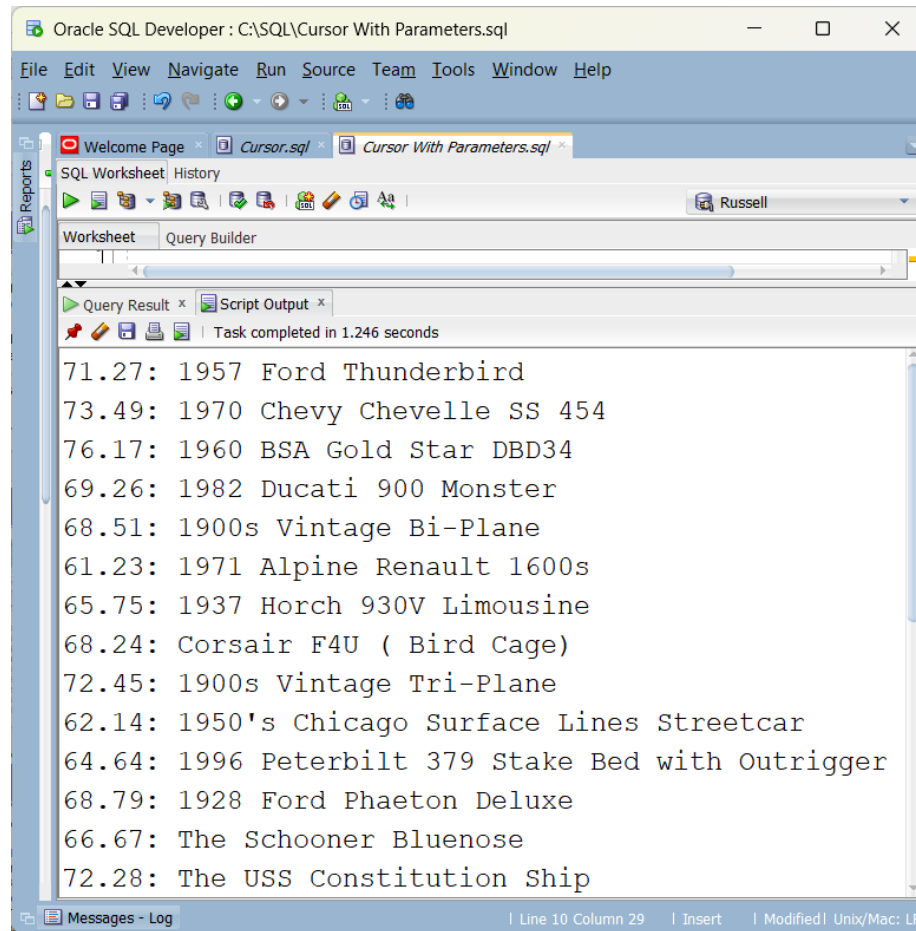
- Can a cursor be flexible, by accepting different parameters each time it is used?
- We have a PRODUCTS table with MSRP amounts.
- One time we can have a temporary result table with prices between 115 and 120
- Another time that cursor will produce a temporary result table with prices between 60 and 80

Explicit Cursor with Parameters

The screenshot displays the Oracle SQL Developer interface. The 'Connections' pane on the left shows a tree of database objects, with 'PRODUCTS' expanded. The 'Script Output' pane at the bottom shows the results of a query, displaying a list of product details. The 'Query Result' pane shows the following data:

Product ID	Product Name
117.44	1952 Citroen-15CV
118.28	1992 Porsche Cayenne Turbo Silver
115.75	Diamond T620 Semi-Skirted Tanker
118.65	ATA: B757-300
118.94	1996 Moto Guzzi 1100i
117.44	1968 Dodge Charger
118.5	1957 Chevy Pickup
115.16	1969 Dodge Charger
116.67	1940 Ford Pickup Truck

Explicit Cursor with Parameters



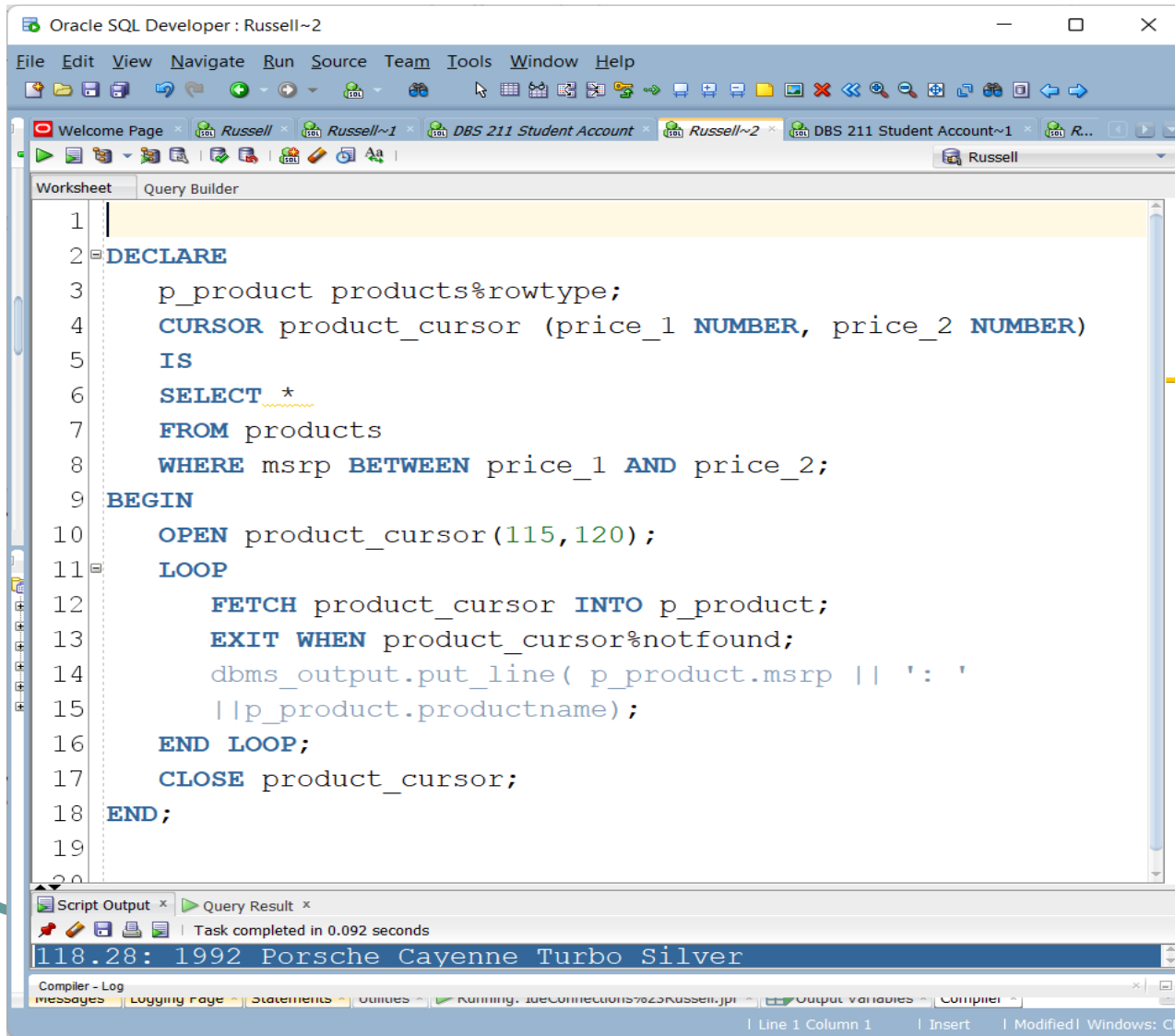
The screenshot shows the Oracle SQL Developer interface. The title bar indicates the file is 'C:\SQL\Cursor With Parameters.sql'. The menu bar includes File, Edit, View, Navigate, Run, Source, Team, Tools, Window, and Help. The toolbar contains icons for file operations, execution, and navigation. The 'SQL Worksheet' tab is active, showing a 'Query Result' window. The 'Query Result' window displays a list of items with their IDs and names. The status bar at the bottom shows 'Task completed in 1.246 seconds' and 'Line 10 Column 29 | Insert | Modified | Unix/Mac: LF'.

```
71.27: 1957 Ford Thunderbird
73.49: 1970 Chevy Chevelle SS 454
76.17: 1960 BSA Gold Star DBD34
69.26: 1982 Ducati 900 Monster
68.51: 1900s Vintage Bi-Plane
61.23: 1971 Alpine Renault 1600s
65.75: 1937 Horch 930V Limousine
68.24: Corsair F4U ( Bird Cage)
72.45: 1900s Vintage Tri-Plane
62.14: 1950's Chicago Surface Lines Streetcar
64.64: 1996 Peterbilt 379 Stake Bed with Outrigger
68.79: 1928 Ford Phaeton Deluxe
66.67: The Schooner Bluenose
72.28: The USS Constitution Ship
```

%Rowtype

- Used to pull in the column type definitions for an entire row of a table
- `variable_name table_or_view_name%ROWTYPE;`
- For every column of the table or view, the record has a field with the same name and data type.
- PRODUCTS has ProductName, MSRP
- `p_product PRODUCTS%ROWTYPE`
- We now have variables called `p_product.productname` and `p_product.msrp`
- Which can be loaded from the table's ProductName and MSRP columns and the column types match

Explicit Cursor with Parameters



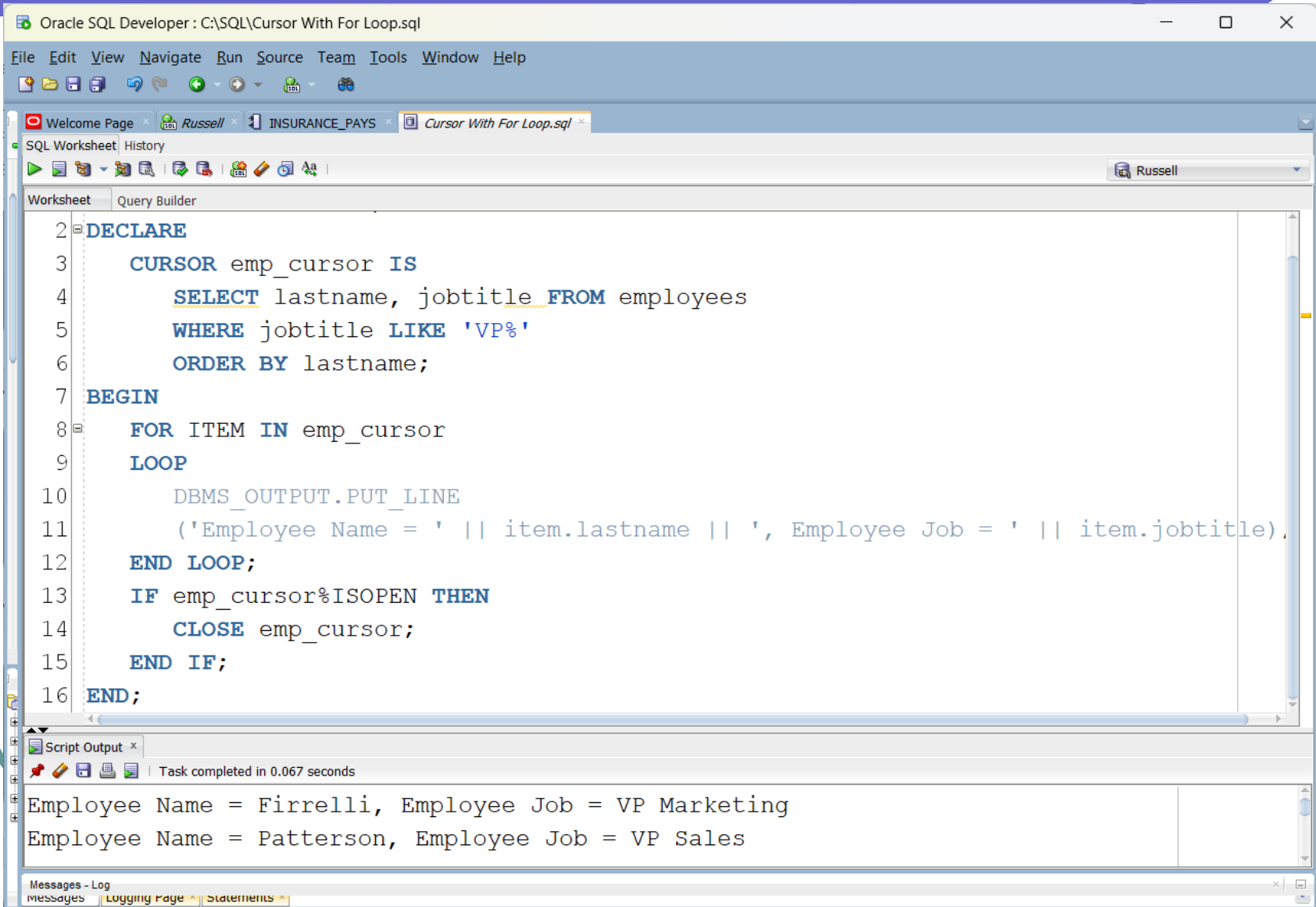
The screenshot displays the Oracle SQL Developer interface. The main window shows a SQL script in the 'Worksheet' tab. The script defines an explicit cursor named 'product_cursor' with two parameters, 'price_1' and 'price_2', both of type 'NUMBER'. The cursor is used in a loop to fetch data from the 'products' table where the 'msrp' is between 'price_1' and 'price_2'. The script is executed, and the 'Script Output' window at the bottom shows the result: '118.28: 1992 Porsche Cayenne Turbo Silver'.

```
1  
2 DECLARE  
3   p_product products%rowtype;  
4   CURSOR product_cursor (price_1 NUMBER, price_2 NUMBER)  
5   IS  
6   SELECT *  
7   FROM products  
8   WHERE msrp BETWEEN price_1 AND price_2;  
9 BEGIN  
10  OPEN product_cursor(115,120);  
11  LOOP  
12    FETCH product_cursor INTO p_product;  
13    EXIT WHEN product_cursor%notfound;  
14    dbms_output.put_line( p_product.msrp || ': '  
15    || p_product.productname);  
16  END LOOP;  
17  CLOSE product_cursor;  
18 END;  
19  
20
```

Script Output x Query Result x
Task completed in 0.092 seconds
118.28: 1992 Porsche Cayenne Turbo Silver
Compiler - Log
Messages | Logging Page | Statements | Variables | Running: jdbc:oracle:thin:@Russell.jpr | Output Variables | Compiler

Line 1 Column 1 | Insert | Modified | Windows: CR

Explicit Cursor in FOR LOOP



The screenshot displays the Oracle SQL Developer interface. The main window shows a SQL script titled 'Cursor With For Loop.sql'. The script defines an explicit cursor named 'emp_cursor' and uses a FOR loop to iterate over the results of a query. The query selects the last name and job title of employees whose job title starts with 'VP'. The output of the script is displayed in the 'Script Output' window, showing two rows of data: 'Employee Name = Firrelli, Employee Job = VP Marketing' and 'Employee Name = Patterson, Employee Job = VP Sales'.

```
2 DECLARE
3     CURSOR emp_cursor IS
4         SELECT lastname, jobtitle FROM employees
5         WHERE jobtitle LIKE 'VP%'
6         ORDER BY lastname;
7 BEGIN
8     FOR ITEM IN emp_cursor
9     LOOP
10         DBMS_OUTPUT.PUT_LINE
11             ('Employee Name = ' || item.lastname || ', Employee Job = ' || item.jobtitle);
12     END LOOP;
13     IF emp_cursor%ISOPEN THEN
14         CLOSE emp_cursor;
15     END IF;
16 END;
```

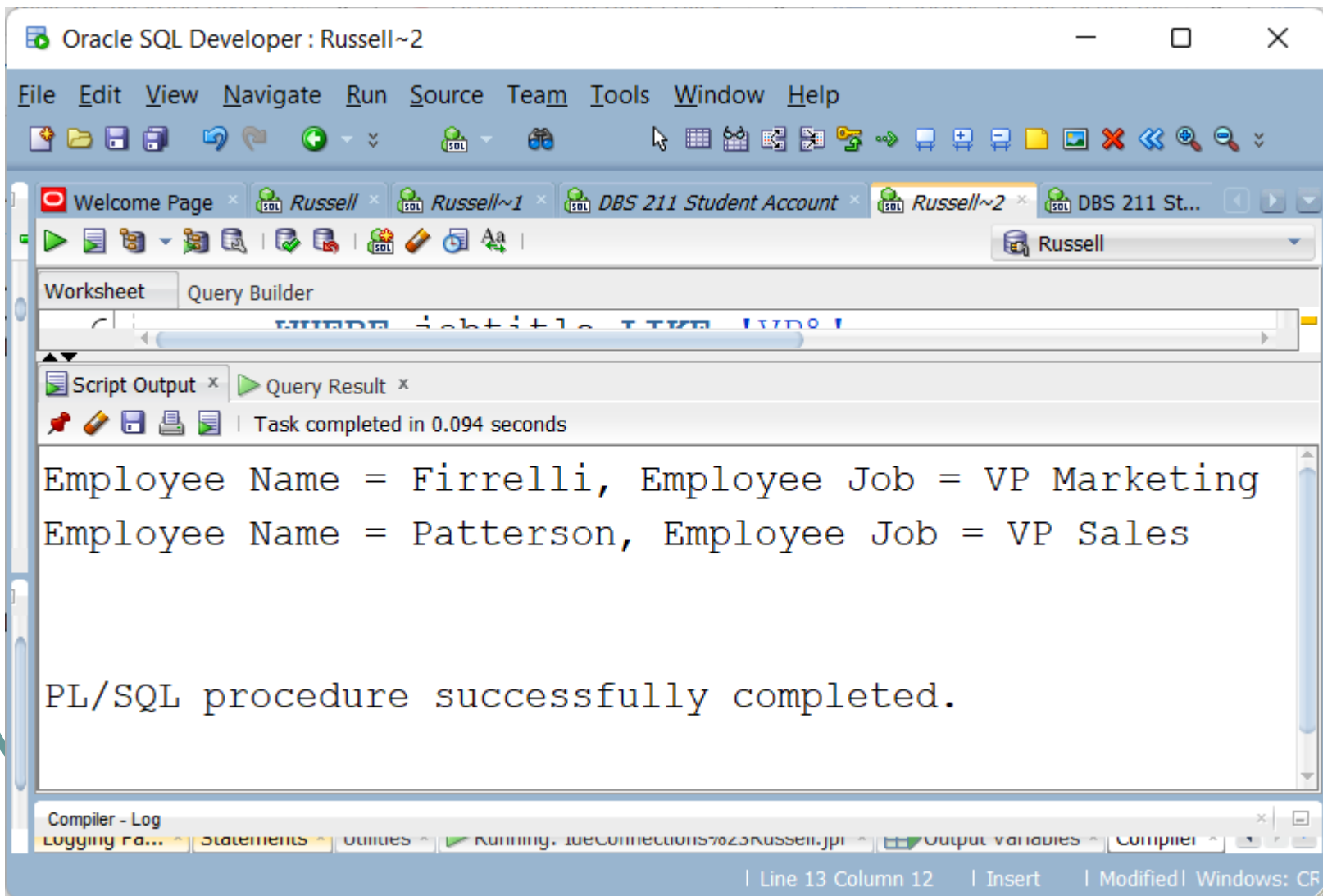
Script Output x

Task completed in 0.067 seconds

Employee Name = Firrelli, Employee Job = VP Marketing
Employee Name = Patterson, Employee Job = VP Sales

Messages - Log
messages | Logging Page | Statements

Explicit Cursor in FOR LOOP



The screenshot displays the Oracle SQL Developer interface. The main window shows a script that has been executed successfully. The 'Script Output' pane at the bottom displays the results of the procedure, which includes two rows of employee data and a confirmation message.

```
Employee Name = Firrelli, Employee Job = VP Marketing  
Employee Name = Patterson, Employee Job = VP Sales  
  
PL/SQL procedure successfully completed.
```

The status bar at the bottom indicates the current position is at Line 13, Column 12, and the window is titled 'Modified | Windows: CR'.

Explicit Cursor Attributes

Attributes	Description
%FOUND	INVALID_CURSOR: if the cursor is not open NULL: before we fetch the first row FALSE: if no row is fetched successfully TRUE: if the row is fetched successfully
%NOTFOUND	INVALID_CURSOR: if the cursor is not open. NULL: before we fetch the first row
%ROWCOUNT	INVALID_CURSOR: if the cursor is not open Otherwise: It returns the number of rows returned from the cursor
%ISOPEN	TRUE: if the cursor is open FALSE: if the cursor is not open

%RowCount

- CursorName%RowCount
- Cursors do not know how many rows there are when the cursor is opened
- The count is incremented with each successful Fetch
- You can use this feature within a loop while fetching rows or after the loop is finished to report on total rows fetched

%RowCount

The screenshot displays the Oracle SQL Developer interface. The main window shows a PL/SQL script in the 'Worksheet' tab, which uses the `%RowCount` attribute to track the number of rows processed by a cursor. The script is as follows:

```
10 OPEN emp_cursor;
11 DBMS_OUTPUT.Put_Line ('Cursor opened and row count is ' || emp_cursor%RowCount );
12 LOOP
13     FETCH emp_cursor INTO e_last_name, e_job_tile;
14     EXIT WHEN emp_cursor%notfound;
15     dbms_output.put_line(e_last_name || ' ' || e_job_tile);
16 END LOOP;
17 DBMS_OUTPUT.Put_Line (emp_cursor%RowCount || ' employees found' );
18 CLOSE emp_cursor;
19 END;
```

The 'Script Output' window at the bottom shows the results of the script execution, which completed in 0.087 seconds. The output is:

```
Cursor opened and row count is 0
Firrelli VP Marketing
Patterson VP Sales
2 employees found
```

The status bar at the bottom indicates the file is saved as 'DBS311 Fall 24 ZB~21' and shows the current cursor position as 'Line 1 Column 1'.

Cursor CountriesCursor is:

Oracle SQL Developer : A DBS311 Student Account F25~2

File Edit View Navigate Run Source Team Tools Window Help

Welcome Page x A DBS311 Student Account F25~2 x CUSTCOUNTRY x

Worksheet Query Builder

```
1 select country, count(*) TOTALCUSTOMERS
2 from customers
3 group by country
4 having count(*) > 7
5 order by country;
```

Query Result x

All Rows Fetched: 3 in 1.858 seconds

	COUNTRY	TOTALCUSTOMERS
1	France	12
2	Germany	13
3	USA	36

Messages - Log

messages Logging Page Statements

| Line 1 Column 40 | Insert | Modified | Windows: CF

Oracle SQL Developer : A DBS311 Student Account F25~2

File Edit View Navigate Run Source Team Tools Window Help

Welcome Page x A DBS311 Student Account F25~2 x CUSTCOUNTRY x

Worksheet Query Builder

```
1 select country, count(*) TOTALCUSTOMERS
2 from customers
3 group by country
4 having count(*) > NUMIN
5 order by country;
```

Query Result x

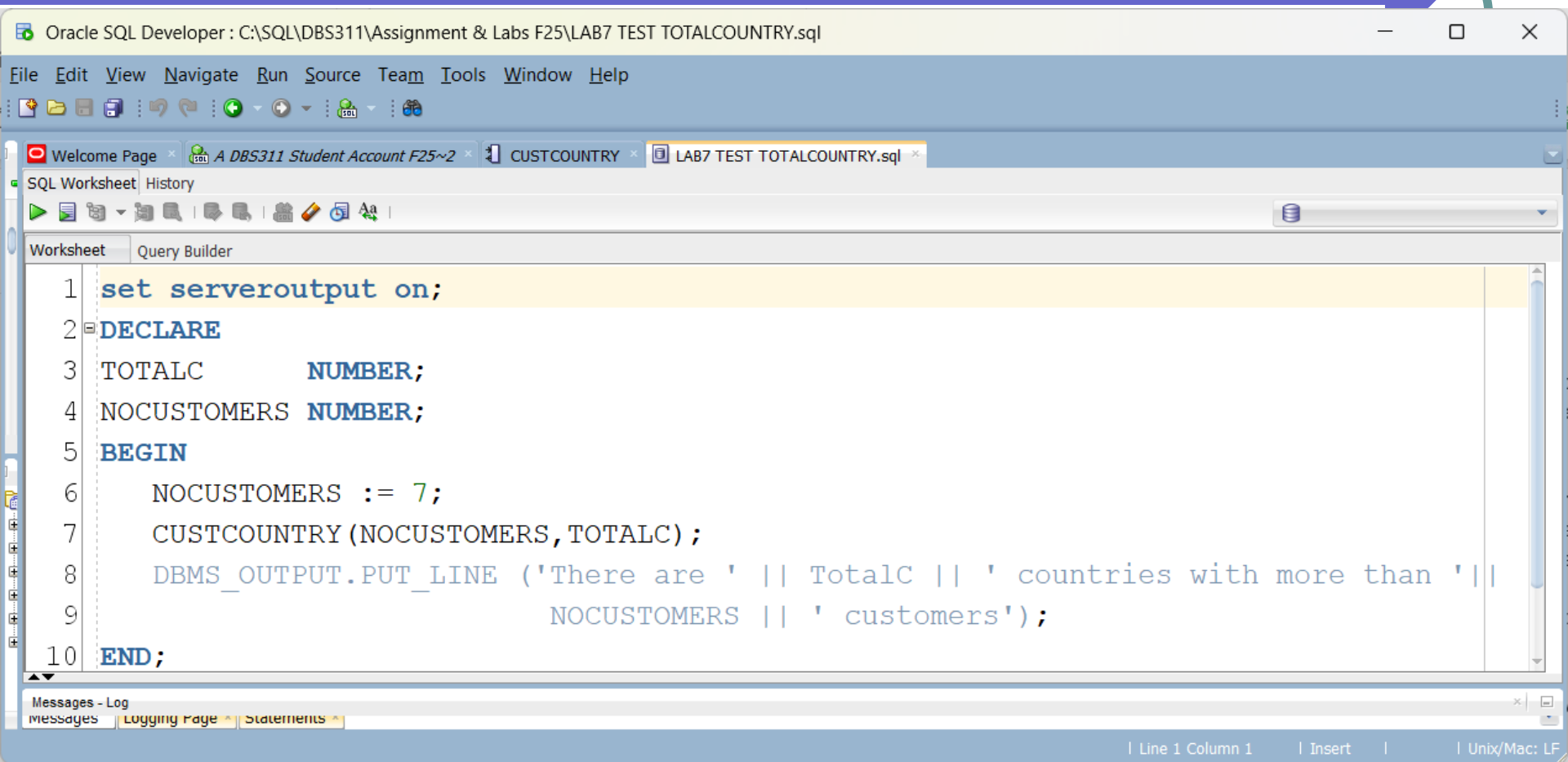
All Rows Fetched: 3 in 1.858 seconds

	COUNTRY	TOTALCUSTOMERS
1	France	12
2	Germany	13
3	USA	36

Messages - Log

messages Logging Page Statements

| Line 4 Column 26 | Insert | Modified | Windows: CF



USER DEFINED FUNCTIONS

- Created with PL/SQL or Java WITH ORACLE
- Created with SQL/PL or RPGLE, C ... programs with DB2
- UDF's can appear in an SQL statement
- A UDF in a SELECT statement can be found:
 - in the select list
 - as the condition in a WHERE clause
 - in an ORDER BY clause
 - in a GROUP BY clause

USER DEFINED FUNCTIONS

- UDF's can appear in other SQL statements
- A UDF in an INSERT statement can be found:
 - in the VALUES clause
- A UDF in an UPDATE statement can be found:
 - in the WHERE clause
- A PL/SQL function is a stored piece of code that can be called from other functions/procedures or programs. A function returns a value using the RETURN clause.

UDF with a passed parameter

The screenshot shows the Oracle SQL Developer interface. The main window displays a SQL query in the Query Builder tab:

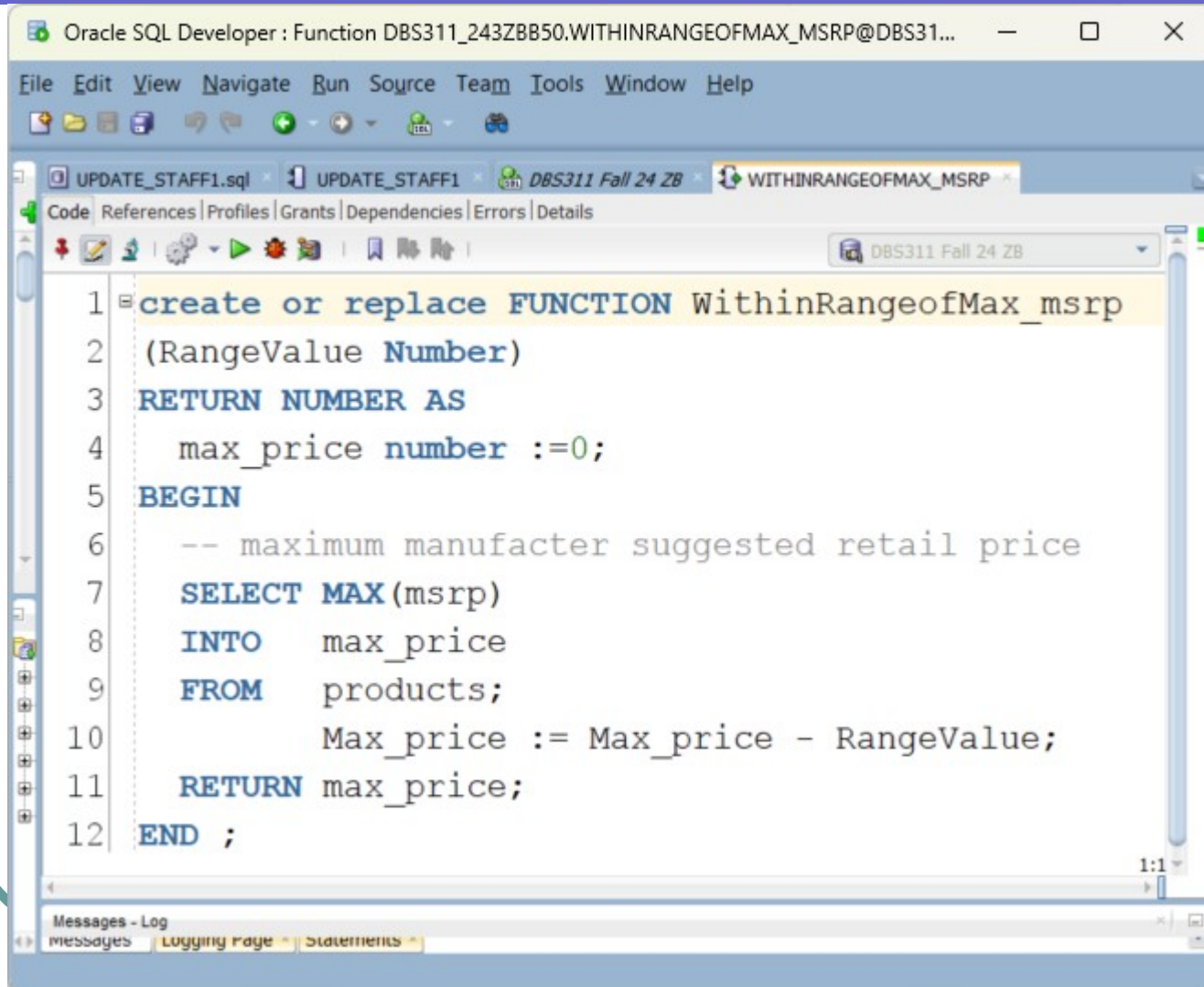
```
1 SELECT PRODUCTNAME, MSRP
2 FROM PRODUCTS
3 WHERE MSRP >= WITHINRANGEOFMAX_MSRP(20);
```

Below the query, the Query Result tab shows the results of the query. The status bar indicates "All Rows Fetched: 3 in 0.036 seconds". The results are displayed in a table with two columns: PRODUCTNAME and MSRP.

	PRODUCTNAME	MSRP
1	1952 Alpine Renault 1300	214.3
2	1968 Ford Mustang	194.57
3	2001 Ferrari Enzo	207.8

The bottom of the window shows the Messages - Log panel with tabs for messages, Logging Page, and Statements. The status bar at the bottom indicates "Line 5 Column 1 | Insert | Modified | Windows: CF".

UDF with a passed parameter



The screenshot shows the Oracle SQL Developer interface. The title bar reads "Oracle SQL Developer : Function DBS311_243ZBB50.WITHINRANGE OF MAX_MSRP@DBS311...". The menu bar includes File, Edit, View, Navigate, Run, Source, Team, Tools, Window, and Help. The toolbar contains icons for file operations, execution, and debugging. The editor window has tabs for "UPDATE_STAFF1.sql", "UPDATE_STAFF1", "DBS311 Fall 24 ZB", and "WITHINRANGE OF MAX_MSRP". The "WITHINRANGE OF MAX_MSRP" tab is active, showing a PL/SQL function definition. The code is as follows:

```
1 create or replace FUNCTION WithinRangeofMax_msrp
2 (RangeValue Number)
3 RETURN NUMBER AS
4   max_price number :=0;
5 BEGIN
6   -- maximum manufacter suggested retail price
7   SELECT MAX(msrp)
8   INTO   max_price
9   FROM   products;
10        Max_price := Max_price - RangeValue;
11   RETURN max_price;
12 END ;
```

The bottom of the window shows a "Messages - Log" panel with tabs for "messages", "Logging Page", and "Statements".

A UDF with two input parameters

The screenshot shows the Oracle SQL Developer interface. The title bar indicates the database is 'DBS311 Fall 24 ZB~15'. The menu bar includes File, Edit, View, Navigate, Run, Source, Team, Tools, Window, and Help. The toolbar contains icons for file operations, execution, and formatting. The 'Worksheet' tab is active, displaying a SQL query in the 'Query Builder' view. The query is as follows:

```
9 SELECT PRODUCTNAME,  
10        BUYPRICE,  
11        MSRP,  
12        PRICEMARKUP (MSRP,BUYPRICE) MARKUP  
13 FROM PRODUCTS;
```

Below the query editor, the 'Script Output' and 'Query Result' tabs are visible. The 'Query Result' tab shows the output of the query, which has fetched 50 rows in 0.009 seconds. The results are displayed in a table with four columns: PRODUCTNAME, BUYPRICE, MSRP, and MARKUP.

	PRODUCTNAME	BUYPRICE	MSRP	MARKUP
1	Boeing X-32A JSF	32.77	49.66	16.89
2	Pont Yacht	33.3	50.31	17.01
3	1969 Harley Davidson Ultimate Chopper	48.81	95.7	46.89
4	1952 Alpine Renault 1300	98.58	214.3	115.72
5	1996 Moto Guzzi 1100i	68.99	118.94	49.95
6	2003 Harley-Davidson Eagle Drag Bike	91.02	193.66	102.64
7	1972 Alfa Romeo GTA	85.68	136	50.32
8	1962 LanciaA Delta 16V	103.42	147.74	44.32
9	1968 Ford Mustang	95.34	194.57	99.23
10	2001 Ferrari Enzo	95.59	195.33	99.74
11	1958 Setra Bus	77.9	136.67	58.77
12	2002 Suzuki XREO	66.27	150.62	84.35
13	1969 Corvair Monza	89.14	151.08	61.94
14	1968 Dodge Charger	75.16	117.44	42.28

A UDF with two input parameters

- create or replace FUNCTION PRICEMARKUP
- (MSRP IN FLOAT,
- BUYPRICE IN FLOAT
-)
- RETURN FLOAT AS
- MARKUP FLOAT;
- BEGIN
- MARKUP := MSRP - BUYPRICE;
- RETURN(MARKUP);
- END;

Oracle SQL Developer : Russell

File Edit View Navigate Run Source Team Tools Window Help

Welcome Page x Russell x PRICEMARKUP x

Worksheet Query Builder

```
1 SELECT PRODUCTNAME,
2     BUYPRICE,
3     MSRP,
4     PRICEMARKUP (MSRP, BUYPRICE) MARKUP
5 FROM   PRODUCTS
6 WHERE  PRICEMARKUP (MSRP, BUYPRICE) > 60
7 ORDER BY PRICEMARKUP (MSRP, BUYPRICE) DESC;
```

Script Output x Query Result x

SQL | All Rows Fetched: 20 in 2.238 seconds

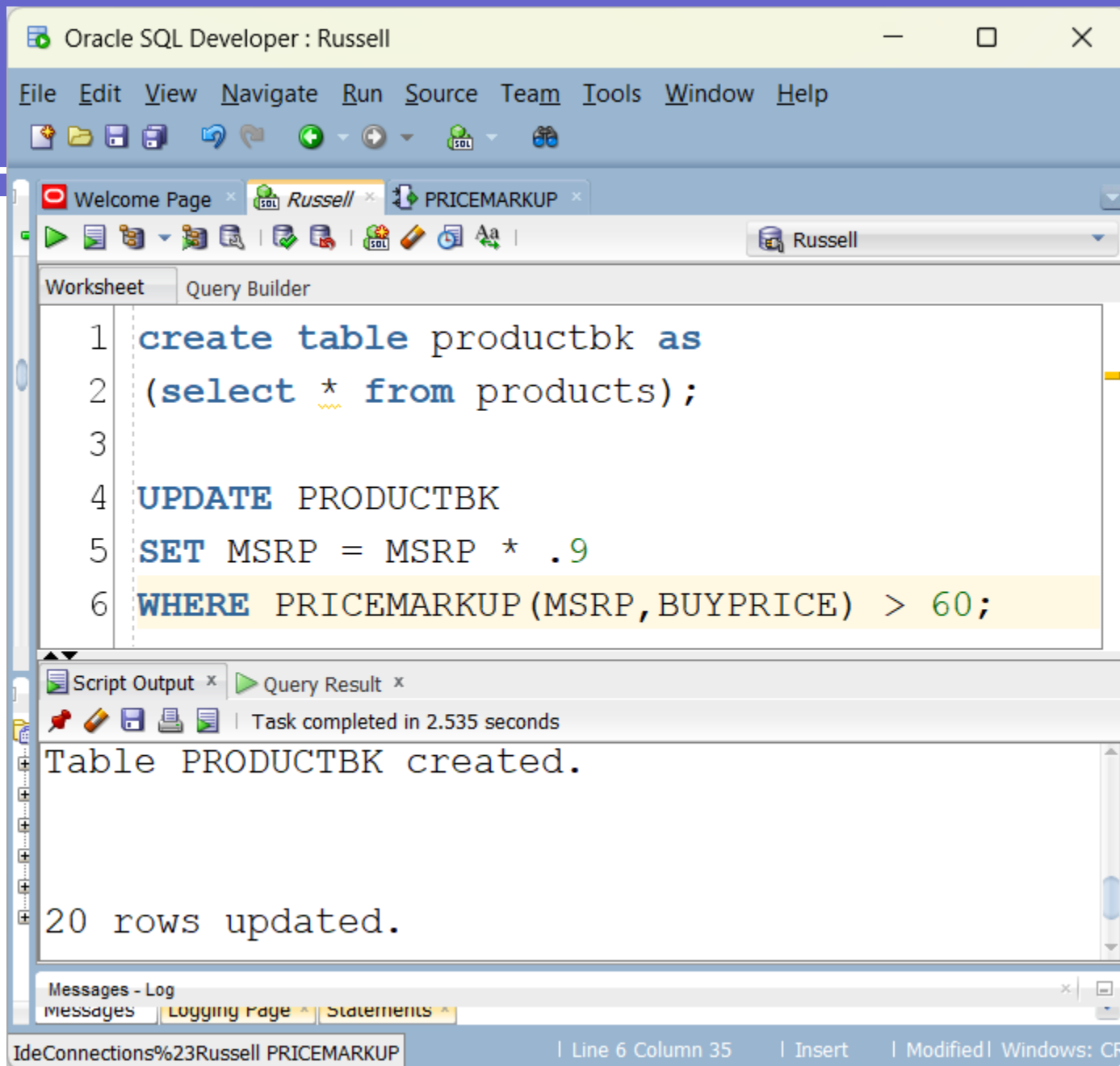
	PRODUCTNAME	BUYPRICE	MSRP	MARKUP
1	1952 Alpine Renault 1300	98.58	214.3	115.72
2	2001 Ferrari Enzo	95.59	207.8	112.21
3	2003 Harley-Davidson Eagle Drag Bike	91.02	193.66	102.64
4	1968 Ford Mustang	95.34	194.57	99.23
5	1928 Mercedes-Benz SSK	72.56	168.75	96.19
6	1992 Ferrari 360 Spider red	77.9	169.34	91.44
7	1969 Ford Falcon	83.05	173.02	89.97
8	2002 Suzuki XREO	66.27	150.62	84.35
9	1917 Grand Touring Sedan	86.7	170	83.3
10	1980s Black Hawk Helicopter	77.27	157.69	80.42
11	1948 Porsche Type 356 Roadster	62.16	141.28	79.12
12	1957 Corvette Convertible	69.93	148.8	78.87
13	1999 Indy 500 Monte Carlo SS	56.76	132	75.24
14	1976 Ford Gran Torino	73.49	146.99	73.5
15	1932 Model A Ford J-Coupe	58.48	127.13	68.65
16	1903 Ford Model A	68.3	136.59	68.29
17	1962 Volkswagen Microbus	61.34	127.79	66.45
18	1957 Chevy Pickup	55.7	118.5	62.8
19	1998 Chrysler Plymouth Prowler	101.51	163.73	62.22
20	1969 Corvair Monza	89.14	151.08	61.94

Messages - Log

er By

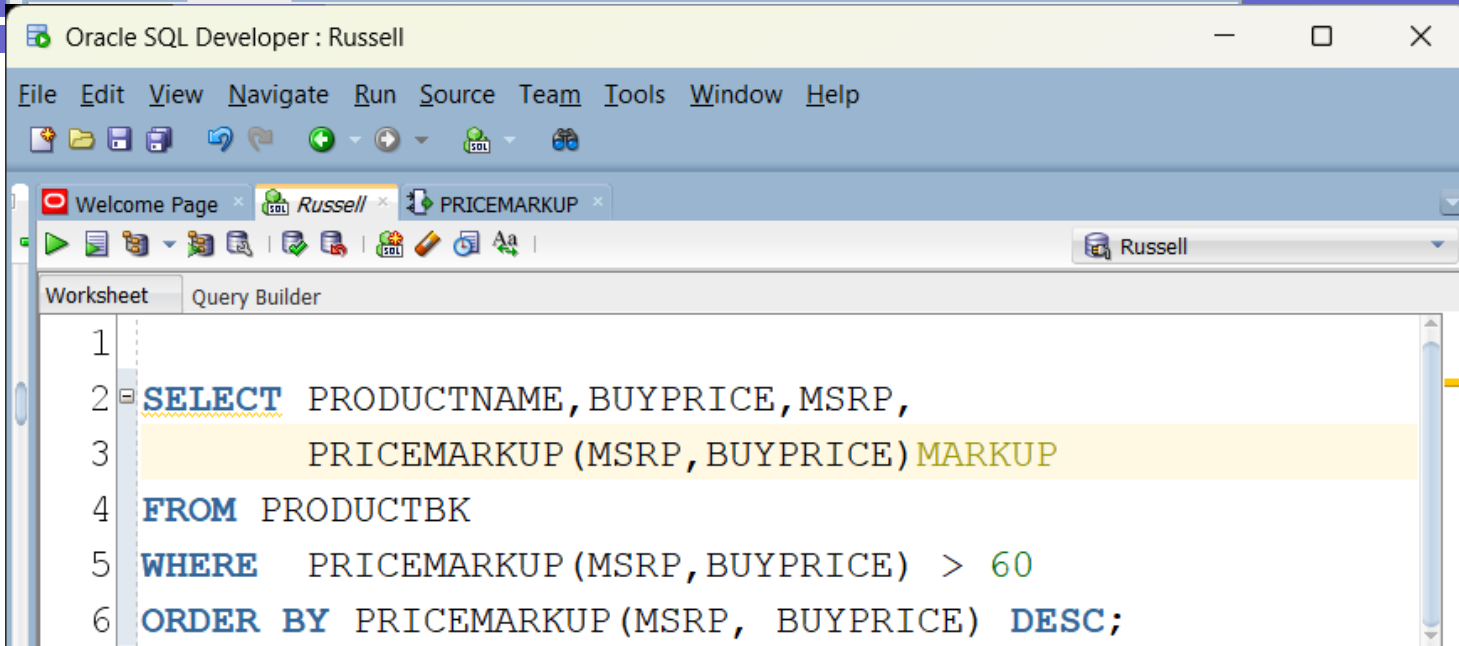
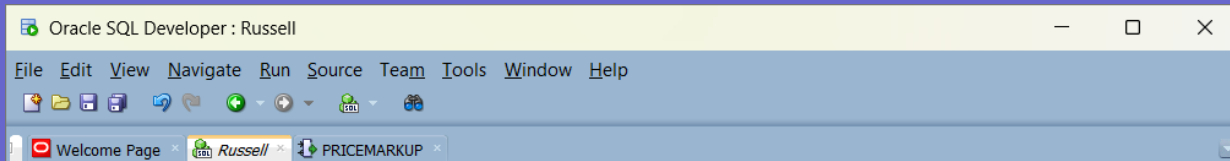
P

SC;



MENT

markup. We can
markup is



Script Output x Query Result x

SQL | All Rows Fetched: 13 in 2.029 seconds

	PRODUCTNAME	BUYPRICE	MSRP	MARKUP
1	1952 Alpine Renault 1300	98.58	192.87	94.29
2	2001 Ferrari Enzo	95.59	187.02	91.43
3	2003 Harley-Davidson Eagle Drag Bike	91.02	174.29	83.27
4	1968 Ford Mustang	95.34	175.11	79.77
5	1928 Mercedes-Benz SSK	72.56	151.88	79.32
6	1992 Ferrari 360 Spider red	77.9	152.41	74.51
7	1969 Ford Falcon	83.05	155.72	72.67
8	2002 Suzuki XREO	66.27	135.56	69.29
9	1917 Grand Touring Sedan	86.7	153	66.3
10	1948 Porsche Type 356 Roadster	62.16	127.15	64.99
11	1980s Black Hawk Helicopter	77.27	141.92	64.65
12	1957 Corvette Convertible	69.93	133.92	63.99
13	1999 Indy 500 Monte Carlo SS	56.76	118.8	62.04

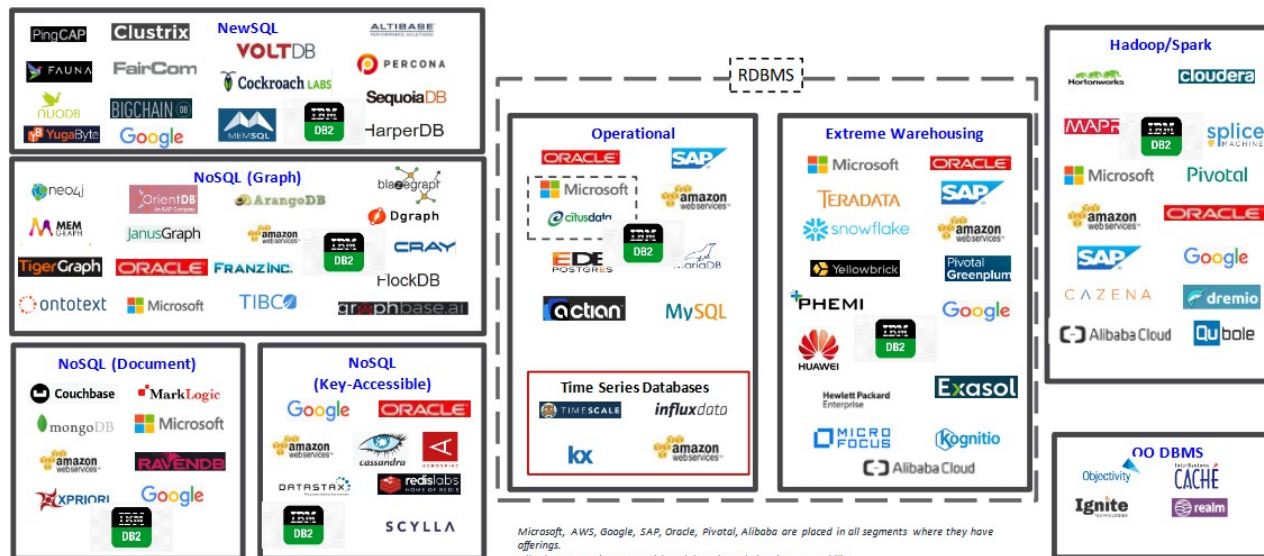
Relational Database Alternatives

- NoSQL
- NewSQL
- Multi Model

Wikipedia Definitions

- NoSQL – “Not only SQL” – A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- NewSQL - is a class of relational database management systems that seek to provide the scalability of NoSQL systems for online transaction processing workloads while maintaining the ACID guarantees of a traditional database system
- Multi-model - Most database management systems are organized around a single data model that determines how data can be organized, stored, and manipulated. In contrast, a multi-model database is designed to support multiple data models against a single, integrated backend

Database Landscape



NoSQL

- A class of database management systems that depart from traditional RDBMSs
- Does not use SQL as the primary query language
- Is “schema-less”
- No rigid schema enforced by the DBMS
- Programmer-friendly for adding fields to a document
- Might not guarantee full `ACID` behavior
- Often has a distributed, fault-tolerant, elastic architecture
- Highly optimized for retrieve and append operations over great quantities of data

ACID Transactions

- **Atomicity** - all operations in a transaction succeed or are all rolled back
- **Consistency** - after transactions are complete, the database remains consistent
- **Isolation** - transactions are executed without interfering with other transactions
- **Durability** - after transactions are completed, changes will be permanent

MongoDB

- What is MongoDB?
- MongoDB is a document-oriented database and differs from a relational one.
- It scales up easier compared to a relational database.
- MongoDB is a powerful, flexible, and scalable general-purpose database.
- It provides the following features:
 - Indexing
 - Aggregations
 - File Storage
 - Special collection types

MongoDB Ease of Use

- The concept of a row is replaced with a document which is more flexible.
- By using documents and arrays, complex hierarchical relationships can be represented with a single record.
- MongoDB is schemaless.
- There is not predefined schema.
- The type and size of a document's keys and values can be variable.
- Add or remove fields is easier.
- Different models can be chosen

MongoDB Easy Scaling

- As data grows at an incredible pace, the databases need to scale up.
- To scale:
 - Large machines can be used to scale up
 - Expensive
 - There is physical limit, more powerful machine may not exist.
 - Partitioning data across more machines can help scale out
 - More storage space by adding servers and computers to your cluster
 - Cheaper
 - But difficult to manage thousands of machines
- MongoDB as a document-oriented model scales out easier by splitting data across multiple servers.

MongoDB Basic Concepts

- Document
 - A document is the basic unit of data
 - A document is equivalent to a row in a relational database
- Collection
 - a collection can be considered as a table but with a dynamic schema
- One MongoDB instance can host multiple collections.

Documents

- Every document has a special key (Id)
- The key of a document is unique within a collection
- Example:
 {"greeting" : "Hello, world!"}
- Key: "greeting"
- Value: "hello, world!"
- A document can contain multiple key/value pairs:
 {"greeting" : "Hello, world!", "foo" : 3}
- Key: "greeting" and value: "hello, world!"
- Key: "foo" and value: 3
- Notice that the type of these two values are different. One is integer and the other one is string.

Document Key

- Document Key
- The type of a key is string (UTF-8 characters).
- The key cannot be the null terminator '\0'.
- Do not include \$ in a key.
- MongoDB is type-sensitive and case-sensitive:
 - `{"foo" : 3}`
 - `{"foo" : "3"}`
- The above documents are distinct.
- The following documents are distinct as well
 - `{"foo" : 3}`
 - `{"Foo" : 3}`
- In Oracle the column empid and Empid are the same
- The columns names are not case-sensitive

Documents

- Duplicate Keys
- A document cannot contain duplicate keys:
 - `{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}`
 - The above document is not a legal document because it has duplicate keys.
 - Will cause an error
 - This is similar to having a two identically named columns in a table row in a relational database

Collections

- A collection is a group of documents.
- A collection in MongoDB can be considered as similar to a table in a relational database.
- The collection has a dynamic schema.
- Documents within a collection can have different schemas.
- ```
{"greeting" : "Hello, world!"}
```
- ```
{"foo" : 5}
```
- The documents have different keys and values and value types
- Two different documents inside the same collection
- A document can be in any collection.
- Can't do the same thing with rows, columns and tables in a relational database

Relational Database

Customer Table

Name	Address	Phone Number
Todd Lynn	90 Park Pl.	(374) 919-8909
Margot Parks	2 Sunset Dr.	(252) 391-3585
Ali Garcia	1902 Windsor St.	(204) 870-7819
Susan Miller	39 Kings Highway	(318) 553-7260

Document Database

Customer Collection

Name:
Todd Lynn

Address:
90 Park Pl.

Phone Number:
(374) 919-8909

Name:
Margot Parks

Address:
2 Sunset Dr.

Phone Number:
(252) 391-3585

Name:
Ali Garcia

Address:
1902 Windsor St.

Phone Number:
(204) 870-7819

Name:
Susan Miller

Address:
39 Kings Highway

Phone Number:
(318) 553-7260

Nomenclatures





SQL	Table	Row	Join
NOSQL	Collection	Document	Embedded Document




Lab

- Lab 7 number 3 asks for you to get started with MongoDB, so you can create a collection and documents based on your assigned database model.
- The MongoDB labs will be done on your pc's or laptops and will require some installation
- The MongoDB labs will be demonstrated online since the schools machines are not set up for this new approach

- Set up an Atlas Account
- Set up Visual Studio Code
- Add the MongoDB extension to VS Code
- Connect from VS Code to Atlas
- Instructions for Atlas and VS Code

Connection String

AutoSave ☐ Off    Connection String • Last Modified: 13 March 

Home Insert Draw Design Layout References Mailings Review View Help  Comments  Editing  Share

Connection String

mongodb Atlas

mongodb+srv://Russ_Pangborn:mypass@cluster0.dyoiaud.mongodb.net/test










user name password database name

can be used to connect from the mongo shell, mongo DB compass, visual studio code or any other application

To locate the connection string in the atlas dashboard - Click on the database link on left hand side

To see the connection string, click on the connect button for cluster 0

You have options to connect to shell, application or MongoDB compass (you want compass)

1 of 1 75 words  English (Canada)  Text Predictions: On  Accessibility: Good to go  Display Settings  Focus     100%

The screenshot shows the MongoDB Atlas web interface with a modal dialog titled "Connect to Cluster0". The dialog has three tabs: "Setup connection security", "Choose a connection method", and "Connect". The "Choose a connection method" tab is active, showing instructions for connecting using VS Code.

Connect to Cluster0

✓ Setup connection security > ✓ Choose a connection method > Connect

Connect Using VS Code

- 1 Install MongoDB for VS Code.**
In [VS Code](#), open "Extensions" in the left navigation and search for "MongoDB for VS Code." Select the extension and click install.
- 2 In VS Code, open the Command Palette.**
Click on "View" and open "Command Palette."
Search "MongoDB: Connect" on the Command Palette and click on "Connect with Connection String."
- 3 Connect to your MongoDB deployment.**
Paste your connection string into the Command Palette.

```
mongodb+srv://Russ_Pangborn:<password>@cluster0.dyoiaud.mongodb.net/test
```

Replace `<password>` with the password for the [Russ_Pangborn](#) user.
When entering your password, make sure all special characters are [URL encoded](#).
- 4 Click "Create New Playground" in MongoDB for VS Code to get started.**
[Learn more about Playgrounds](#)

Having trouble connecting? [View our troubleshooting documentation](#)

The background interface shows the "Database I" page for "Cluster0" with a sidebar containing "DEPLOYMENT", "DATA SERVICES", and "SECURITY". The "Data Lake" section is highlighted with a "PREVIEW" button. The "Data Size" is 155.3 KB, and the "Last 3 days" usage is 512.0 MB.

Temporary IP address when at campus 0.0.0.0

The screenshot shows the MongoDB Atlas interface for a cluster named 'Seneca Coll...'. The 'Network Access' tab is selected, displaying the 'IP Access List'. A notification at the top states: 'We are deploying your changes (current action: configuring MongoDB)'. Below this, the breadcrumb 'SENECA COLLEGE > PROJECT 0' is shown. The 'Network Access' title is prominently displayed. Under the 'IP Access List' tab, a yellow warning box states: 'You will only be able to connect to your cluster from the following list of IP Addresses:'. Below the warning is a table with the following data:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		Active	EDIT DELETE

At the top right of the table area is a '+ ADD IP ADDRESS' button. The left sidebar contains navigation links for Overview, DATABASE, Clusters, SERVICES, Atlas Search, Stream Processing, Triggers, Migration, Data Federation, Data API, SECURITY, Quickstart, Backup, Database Access, Network Access (highlighted), and Advanced. The bottom of the page shows 'System Status: All Good' and copyright information for MongoDB, Inc.

MongoDB Group Database Collections

- Each group needs to select what their Lab database Collections are going to be modeled on
- Tired of books, movies, games as student selections. I am doing audiobooks mainly.
- Email me a list starting from most desirable to least desirable
- I will go in order of emails arrival and inform the group what their databases should model
- If I didn't award your first item – someone beat you to it.

List For your MongoDB

- 1 Lumber
- 2 Flowers
- 3 Guns
- 4 Health Foods
- 5 Tea
- 6 Office Furniture
- 7 Dishwashers
- 8 Tools
- 9 Winter Coats

Lab 7 - 10

- Lab 7 will be marked in class
- Labs 8 – 10 will be run online (not in the classroom)
Lectures still on campus
- The Mongo Labs 8 – 10 will be group labs using your assignment group.

Remaining Schedule

Week	Lecture	Lab
Nov 3 WK	Lecture on Cursors, Functions, MongoDB	Lab 6 due
Nov 10 WK	MongoDB Query	Oracle Lab 7 due
Nov 17 WK	MongoDB Update Documents	Visual Code Lab 8 Due online
Nov 24 WK	Aggregation	Visual Code Lab 9 due online
Dec 1 WK	Review for Final	Assign2 due online
Dec 8 WK	Final Test	Lab 10 due online