

WEEK- 3

# Structured Query Language(SQL)

# Agenda

- ▶ SQL and SQL Standardization
  - ▶ The basic commands and functions of SQL
  - ▶ How to use SQL to query a database to extract useful information (The SELECT statement)
- ▶ Sub-Languages of SQL
- ▶ The Basics of SELECT
- ▶ Aliases (Field and Table Aliases)
- ▶ INSERT - Inserting New Data Rows
- ▶ UPDATE - Updating Existing Data
- ▶ DELETE - Deleting Existing Data

# Introduction to SQL

**SQL:** Structured Query Language

- ▶ Pronounced Sea - Quel

**Universal Language** used specifically for communicating with databases

SQL functions fit into **three broad categories:**

- ▶ DDL - Data definition language
- ▶ DML - Data manipulation language
- ▶ TCL - Transaction Control Language

# What Can SQL do?

SQL can:

- ▶ execute queries against a database
- ▶ retrieve data from a database
- ▶ insert records in a database
- ▶ update records in a database
- ▶ delete records from a database
- ▶ create new databases
- ▶ create new tables in a database
- ▶ create stored procedures in a database
- ▶ create views in a database
- ▶ set permissions on tables, procedures, and views

# SQL - sub-categories

## DDL - Data Definition Language

- ▶ Create database objects such as tables
- ▶ Commands to define access rights to those database objects
- ▶ Essentially working with the **STRUCTURE** of the database design

## DML - Data Manipulation Language

- ▶ Commands to work with the **DATA** stored in the database
- ▶ Includes commands to insert, update, delete, and retrieve data within the database tables objects

## TCL - Transaction Control Language

- ▶ Includes commands to ensure the integrity of the database
- ▶ Working with Multi-Step procedures to ensure everything that is supposed to happen, actually happens.

## PL/SQL      **Procedural Language extensions to SQL**

PL/SQL contains many of the operations that we would find in most programming languages, such as loops, functions, variables, conditional statements, etc.

# The SELECT command

6

Introducing the SELECT command general syntax

```
SELECT <comma separated field list *>  
FROM <tablename(s) *>  
WHERE <condition(s) using logical operators>  
ORDER BY <comma separated field list> ;
```

\* - means required part

# The **SELECT** command **Order** **of Execution**

FROM

WHERE

SELECT

ORDER BY

# Sample Table: Movies

Movies				
Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115



# Listing Table Rows

At a minimum, must specify what you want to select and where you want to select it from

```
SELECT id, title,  
director  
FROM movies;  
  
SELECT * FROM movies;
```

# Aliases (Field and Table Aliases)

*Having the name of columns different than the field name in the output.  
If the output is a new calculated value, it needs an appropriate name  
when joining multiple tables, the same field name may exist in more  
than one table, so we need an identifier*

- ▶ *can be used to save significant typing and make the SQL code much cleaner and easier to read*

# Quotations in Oracle SQL

*Single Quotes: are used to defined string values. For example: Inserting a person's name, would require the name to be in single quotes 'Name'*

*Double Quotes: are used to define or use an object. For example: If a column name has a space in it, which is bad practice, you would have to use double quotes around the name to use it or define it. Aliases can have spaces in them, if they are to be in a printed report, but are required to be defined using double quotes.*

# Field Aliases in Oracle SQL

```
SELECT firstname AS first, lastname AS last  
FROM employees;
```

FIRST	LAST
-------	------

```
SELECT firstname || ' ' || lastname AS name  
FROM employees;
```

NAME
------

```
SELECT productCode, price, quantityordered AS quantity, price * quantityordered  
FROM orders;
```

PRODUCTCODE	PRICE	QUANTITY	[PRICE * QUANTITYORDERED]
-------------	-------	----------	---------------------------

# Table Aliases in Oracle SQL

- ▶ Aliases can also be applied to data sources as well, regardless if they are tables, views, or user-defined objects.
  - ▶ `SELECT firstname, lastname FROM employees e WHERE employeeid = 123;`
- ▶ Table aliases can then be used in other parts of the statement to refer to fields more directly
  - ▶ `SELECT e.firstname, e.lastname FROM employees e WHERE employeeid = 123;`

# Limiting Results and Paging

Sometimes data result sets are very large, but we might only be interested in the last few records, or the first few records.

We can use the OFFSET and FETCH commands in Oracle or the TOP command

in MS SQL to determine which rows in a set are to be returned. If we wanted to see the first 10 orders in the database that were received we could write:

```
SELECT * FROM orders ORDER BY  
orderdate FETCH NEXT 10 ROWS ONLY;
```

# Paging

If we want to skip a few records to see only some in the middle somewhere, we can use the OFFSET clause.

```
-- Page 3 of results with 10 records per page SELECT * FROM orders ORDER BY  
orderdate OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;
```

## ROWNUM (Oracle Only)

Rownum is a builtin value in all query results that indicates the Row Number of the results. Therefore you can use ROWNUM to limit the number of rows returned or return exact rows.

```
SELECT * FROM orders WHERE rownum <= 5; -- returns the first 5 rows  
SELECT * FROM orders WHERE rownum BETWEEN 10 and 20 -- returns the row 10 through 20  
(11 rows)
```

## Single-Line Functions and Calculated Values

In SQL, it is almost always required to manipulate the data to calculate new values. This is how information can be retrieved from data. Therefore, calculations are performed in most SQL SELECT statements and are a basic feature that learners need to understand.

```
-- example  
SELECT  
    productCode,  
    price,  
    quantityOrdered AS quantity,  
    price * quantityOrdered AS subtotal  
FROM orders;
```

PRODUCTCODE	PRICE	QUANTITY	SUBTOTAL
123	1.25	3	3.75
456	5.95	2	11.90

The column defined as subtotal does not exist in the source table. It is a calculated value. The output may look something like this



## Single-Row Functions

- ▶ Numeric Functions are functions involved in mathematical calculations. Some of the most common include ROUND, SQRT, and MOD
  - ▶ `SELECT studentID, mark, maxmarks,  
round(100 * mark / maxmarks,2) AS grade  
FROM studentMarks`

STUDENTID	MARK	MAXMARKS	GRADE
1	27	53	45.28
2	33	53	62.26
3	48	53	90.57

# Date Functions

Date Functions are used to manipulate dates and perform calculations with respect to dates. The formatting of dates are amongst the most common, but also age, length of time etc are also important.

Some important data functions in Oracle include TO\_DATE, MONTHS\_BETWEEN, ADD\_MONTHS, NEXT\_DAY, SYSDATE, and EXTRACT.

```
SELECT sysdate FROM dual; --Today's date from the system
```

```
SELECT  
Months_Between(to_date('05082020','mmddyyyy'),  
to_date('09222020','mmddyyyy')) AS NumMonths  
FROM dual;
```

would output the value -4.4516129032258064516129032258064516129.

```
SELECT  
  firstname,  
  birthDate,  
  EXTRACT(month FROM birthDate) as Mon,  
  EXTRACT(year FROM birthDate) as YR  
  EXTRACT(day FROM birthDate) as Dy  
FROM employees;
```

FIRSTNAME	BIRTHDATE	MON	YR	Dy
John	12-Feb-72	2	1972	12
Mary	23-Sep-76	9	1976	23

# String Functions

▶ `SELECT` `firstname`, `lastname`,  
`CONCAT(firstname, lastname)` `AS` `name`,  
`UPPER(firstname)` `as` `first`,  
`LENGTH(lastname)` `as` `lenlast`,  
`SUBSTR(lastname, 2, 3)` `as` `2l`  
`FROM` `students`;

FIRSTNAME	LASTNAME	NAME	FIRST	LENLAST	2L
John	Smith	JohnSmith	JOHN	5	mit
Mary	Johnston	MaryJohnston	MARY	8	ohn

## Selecting Rows with Comparison Operators

# Filtering Data (WHERE)

```
SELECT firstname, birthDate  
FROM employees  
WHERE EXTRACT(month FROM birthDate) = 5;
```

# Comparison Operators


TABLE 6.7 COMPARISON OPERATORS

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

# Selecting Rows with Comparison Operators

Another Example:

```
SELECT *  
FROM part  
WHERE part_number = 'AX12';
```



Note criteria is in single quotes  
PART\_NUMBER is a character field

**RESULT** - Returns **all** fields but only the single record where the part\_number field matches the given criteria

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
AX12	Iron	104	HW	3	23.95

# Sorting Output

Data is displayed in the order which it was added to the tables initially

- ▶ **Not always true** - some DBMS's will sort tables by their primary key (PK) automatically if no sorting criteria was specified.
  - ▶ Primary Keys are also automatically indexes (more on indexes later)
- ▶ You can specify the **direction of the sorting** by using **ASC** or **DESC**.
  - ▶ ASC is the default direction and is therefore optional
  - ▶ to sort data in descending order, use the **DESC** keyword after each field specified in the **ORDER BY** clause that is to be displayed in descending order

# Sorting Output

To change the order the data is displayed in, use the `ORDER BY` clause in the `SELECT` statement:

```
SELECT *  
  FROM part  
 WHERE on_hand > 90  
 ORDER BY on_hand;
```

**RESULT** - Returns **all** fields but only those records where `on_hand > 90`

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
BH22	Cornpopper	95	HW	3	24.95
AX12	Iron	104	HW	3	23.95
CX11	Blender	112	HW	3	22.95



# Sorting Output - Multiple Columns

You can sort by more than one column

- ▶ The second column will ONLY be sorted If and Only If the first column has duplicates and then only those duplicates are sorted

```
SELECT *  
FROM part  
ORDER BY class, on_hand;
```

RESULT

PART_NUMBER	PART_DESC	ON_HAND	CLASS	WAREHOUSE	PRICE
BT04	Gas Grill	11	AP	2	150
BZ66	Washer	52	AP	3	400
CA14	Griddle	78	HW	3	39.99
BH22	Cornpopper	95	HW	3	24.95
AX12	Iron	104	HW	3	23.95
CX11	Blender	112	HW	3	22.95
AZ52	Dartboard	20	SG	2	12.95
BA74	Basketball	40	SG	1	29.95
CB03	Bike	44	SG	1	300
CZ81	Treadmill	68	SG	2	350

# SQL Comparison Operators

## WHERE

- ▶ In/ not in
  - ▶ Determines if the value of a single field is in a provided comma separated list
- ▶ Between/ not between
  - ▶ A range of data
- ▶ Like / not like
  - ▶ Activates the ability to use wildcards and patterns
- ▶ Is null /Is not null
  - ▶ Required fields vs. optional fields

# SQL Comparison Operators

IN / NOT IN

- Determines if the value of a single field is in a provided comma separated list

## SAMPLES

```
SELECT *  
  FROM part  
 WHERE class IN('HW', 'AP');
```

```
SELECT *  
  FROM part  
 WHERE warehouse IN(1, 2);
```

# SQL Comparison Operators

BETWEEN / NOT BETWEEN

- ▶ Determines if the value of a single field is within a range provided
- ▶ Be careful with inclusive vs exclusive
  - ▶ Suggested to use some practise data to determine if the values given are included or excluded from the range

## SAMPLES

```
SELECT *  
FROM part  
WHERE price BETWEEN 10 AND 50;
```

```
SELECT *  
FROM part  
WHERE part_number BETWEEN 'BA' AND 'CA';
```

# SQL Comparison Operators

IS NULL / NOT IS NULL

- Determines if the value of a single field is NULL

## SAMPLES

```
SELECT *  
  FROM part  
 WHERE price IS NULL;  
-- this returns prices that need to still be entered
```

```
SELECT *  
  FROM part  
 WHERE price IS NOT NULL;  
-- this returns only those products where the price  
has been entered
```

# SQL Comparison Operators

LIKE / NOT LIKE

► Is a comparator that allows the use of wildcards

## SAMPLES

```
SELECT *  
  FROM part  
 WHERE class LIKE 'HW';  
-- this is the same as class = 'HW' but now can use wildcards
```

```
SELECT *  
  FROM part  
 WHERE part_desc LIKE 'B%';  
-- this returns all products whose descriptions start with a capital B
```

# Wildcards

## String Wildcards

- ▶ Allow scripting when part of the required strings are unknown
  - ▶ Examples: starts with, ends with, contains
  - ▶ Use the % wildcard as a placeholder of unknown characters and unknown length

```
SELECT *  
  FROM part  
 WHERE part_desc LIKE 'B%';  
-- this returns all products whose descriptions start with a capital B and any number of  
characters afterwards
```

# Wildcards continued

```
SELECT *  
FROM part  
WHERE upper(part_desc) LIKE '%D';
```

**Returns all rows where the part description ends with the letter 'D'.**

- Note the use of the single function `upper()`. Strings are case sensitive in SQL and therefore we must control the statement as we can not assume the data in the database was entered in any specific way



# Wildcards continued

```
SELECT *  
  FROM part  
 WHERE lower(part_desc) LIKE '%pop%';
```

**Returns all rows where the part description contains the phrase 'POP' within it (at any location, including start, middle and end).**

- Note the use of the single function `lower()`. Strings are case sensitive in SQL and therefore we must control the statement as we can not assume the data in the database was entered in any specific way.

# CRUD

CRUD is a term most programmers should become familiar with

C - Create (`INSERT`)

R - Read (`SELECT`)

U - Update (`UPDATE`)

D - Delete (`DELETE`)

This term is used consistently throughout the database and programming industries.

These are DML statements.

# INSERT

Inserting **NEW** records (rows) into a database

- ▶ There are a few different ways to do insert statements, for now we will cover just 2
  - ▶ We will ignore tables with automatically generated primary keys for now

## Syntax 1

```
INSERT INTO <table name>  
(<comma separated field list>  
VALUES  
(<comma separated value list>;
```

- ▶ In this syntax, the order and inclusion of fields are optional
  - ▶ Order of fields and values must match
  - ▶ All Required fields must be included
- ▶ Syntax 2

```
INSERT INTO <table name>  
VALUES  
(<comma separated value list>;
```

# INSERT

## Samples

```
INSERT INTO parts  
  (part_desc, part_number, on_hand, class, warehouse, price)  
VALUES  
  ('Soccer Ball', 'RC16', 26, 'SG', 1, 24.95);
```

```
INSERT INTO parts  
VALUES  
  ('RC16', 'Soccer Ball', 26, 'SG', 1, 24.95);
```

**Note: the order of the fields**

# Inserting Multiple Rows in One Statement

- ▶ INSERT ALL

```
INTO <tablename> VALUES ( <value list comma separated >)
```

```
INTO <tablename> VALUES ( <value list comma separated >)
```

```
INTO <tablename> VALUES ( <value list comma separated >)
```

```
SELECT * FROM DUAL;
```

```
SELECT <some field name> FROM <table name>;
```

- ▶ -- note the field and table names in the SELECT do NOT need to from the same table.

- ▶ -- Example:

- ▶ INSERT ALL

```
INTO offices VALUES (8, 'Toronto', '+1 416 555 1111', '200 Young St. N., NULL, 'ON', 'Canada', 'M4A3A1', 'NA)
```

```
INTO offices VALUES (9, 'Oshawa', '+1 905 555 2222', '155 Simcoe. S., NULL, 'ON', 'Canada', 'N2L3G4', 'NA)
```

```
INTO offices VALUES (10, 'Montreal', '+1 268 555 3333', '1245 Rue Lavac, NULL, 'QC', 'Canada', 'K3S2H4', 'NA)
```

- ▶ SELECT \* FROM offices;

# Inserting Multiple Rows From another Table

```
INSERT INTO <tablename> (<fieldlist>) SELECT <fieldlist> INSERT INTO  
<tablename> (<fieldlist>)  
  SELECT <fieldlist>  
  FROM <tablename>  
  WHERE <condition>      -- optional  
  ORDER BY <fieldlist>;  -- optional
```

- ▶ the fieldlist in the SELECT must match the fieldlist in the INSERT (not same names, but same meaning)
- ▶ the order of the two fieldlists (SELECT and INSERT) must be the same
- ▶ you can choose not to include the fieldlist in the INSERT line, but then must have all fields in the right order in the SELECT line
- ▶ The WHERE clause is optional
- ▶ The ORDER BY is also optional and not typically needed unless there are autonumbering sequences involved

# UPDATE

To update already **EXISTING** records (Rows)

- ▶ Only include those fields that are changing
- ▶ Make sure you have a WHERE clause that specifies EXACTLY what records are to be updated.

## GENERIC SYNTAX

```
UPDATE <table name> SET  
    <field1>=<value1>,  
    <field2>=<value2>,  
    ....  
WHERE  
    <conditional statement>;
```

# UPDATE continued

## Samples

Updates a single record and changes the description

```
UPDATE parts SET  
    part_desc = 'Gas/Propane Grill'  
WHERE part_number = 'BT04';
```

Updates ALL records matching the WHERE criteria and increases the price by 10%

```
UPDATE parts SET  
    price = price * 1.10  
WHERE class = 'SG';
```



# DELETE

Removes data from the table

- ▶ Most of the time, this is not easily undone (be very careful)
- ▶ 99% of the time a WHERE clause is used to specify the EXACT records to be deleted

## GENERIC SYNTAX

```
DELETE FROM <table name>  
WHERE  
<conditional statement>;
```

# DELETE continued

## Samples

Deletes a single record from the parts table (because PK is specified)

```
DELETE FROM parts  
WHERE part_number = 'BT04';
```

Deletes ALL records matching the WHERE criteria - oops maybe

```
DELETE FROM parts  
WHERE warehouse = 2;
```