# **COMP-1701 - Transferring Data to Databases**

## **Data Science & Machine Learning (DSML) - RRC Polytech**

Module C.1 - Database Basics - SELECT, Data Types, and Conditionals

Using the SELECT commands, we can get a simple understanding of all the basic data types and conditional comparison operations.

**SELECT** column1, column2 ... , columnN   
**FROM** table\_view\_subquery\_temp\_result\_set   
**WHERE** filter\_conditional;

The **SELECT** query returns a ***Temporary Result Set***, using a **FROM** clause to pull **column** and **row** data from a:

* **TABLE**
* **VIEW** (which is a database stored SELECT query), or
* **SUBQUERY** (which is a temporary result set that is ran within another SELECT query, as you will see shortly)

So you should be able to read this as:

* **SELECT** the ***COLUMNs*** (comma delimited)
* **FROM** the **TABLE**, **VIEW**, and/or **SUBQUERY**
* and typically using a filtering **WHERE** clause which is conditional statement that must evaluate to **TRUE** (ie: 1=1) for **rows** to be returned

Data Types

Data types can be classified in 4 main groupings: Boolean, Numeric, Strings, and Date Time fields

* **Boolean** - **BIT** field - TRUE(=1), FALSE(=0) values

-- BIT/boolean is actually a TINYINT(1) or  
-- INT(1) only accepting 0 or 1 (FALSE or TRUE)  
-- as values

SELECT FALSE AS b0, TRUE AS b1;  
  
-- FALSE returns 0  
-- TRUE returns 1  
-- b0 and b1 are called labels  
-- the results returned after executing this   
-- SQL query are a result set

-- using the temp result set as a subquery  
-- in the FROM to do some conditionals in   
-- the SELECT  
--  
SELECT *trs*.b0, b1  
 , b0=b1, b0<>b1, b0!=b1, b0=b0  
 , trs.b1=TRUE  
FROM (SELECT FALSE AS b0, TRUE AS b1) *trs*;  
  
-- trs needs to be specified here, this is a  
-- *table alias*, it's an abbreviation of your  
-- choice, I used trs to abbreviate temp result  
-- set and also can be used on the columns in  
-- the SELECT clause  
--   
-- So FROM trs, we are calling columns  
-- b0 and b1, to display them, and compare them  
-- via the:  
--   
-- b0=b1 returns 0 (FALSE) as TRUE equal to   
-- FALSE is FALSE  
--  
-- b0<>b1 and b0!=b1 returns 1 (TRUE) as they do  
-- not equal each other as FALSE does not equal  
-- TRUE  
--  
-- b0=b0 returns 1 as b0 compared to itself with  
-- values of FALSE are equal   
-- equal TRUE  
--  
-- trs.b1=TRUE, where we use an alias-column  
-- reference, that calls b1 from trs and then  
-- comparing it to the value of TRUE, which  
-- returns 1, as it b1 does equal TRUE

* **Numerics:**
* Integers - **INT(4)** - 0,1,2,3,...
* Decimals – NUMERIC(s,p) or DECIMAL(s,p), **FLOAT**(s,p), **DOUBLE**(s,p):  
    
  1.1 -> NUMERIC(2,1)   
  2.22, -> NUMERIC(3,2)   
  33.333 -> NUMERIC(5,3)  
  444.444 -> NUMERIC(6,3)
* specifying the size (s) and number of decimal places or what is referred to as the precision (p)
* the sign of the number and the dot are not counted within the size and precision
* Each have a cap to how low and high a number can be stored
* Numbers can have **SIGNED** or **UNSIGNED** specifications, which indicate whether the values in the columns are **positive only** or can be **negative and positive**, respectively (as mentioned, the sign is not counted within the size and precision)

-- returns for columns, w/ labels-values listed  
  
SELECT 1 AS i1, 2 AS i2  
 , 1.0 AS d1, 2.22 AS d2  
  
-- again, taking this SELECT and using it as a  
-- temporary result set  
  
SELECT i1, i2, d1, d2 -- returns: 1, 2, 1.0, 2.22  
 , i1<i2 -- returns 1 as 1<2 is TRUE  
 , i1<=d1 -- returns 1 as 1<=2 is TRUE  
 , i1=d1 -- returns 1 as 1=1.0 is TRUE  
 , i1!=d1 -- returns 1 as 1!=1.0 is FALSE  
 , i2>i1 -- returns 1 as 2>1 is TRUE  
 , d2>=i2 -- returns 1 as 2.22>=2 is TRUE  
 , 1.5 BETWEEN i1 AND d2 -- 1<=1.5<=2.22 TRUE  
 , 2.22 BETWEEN i1 AND d2 -- 1<=2.22<=2.22 TRUE  
 , 3 BETWEEN i1 AND d2 -- 1<=3<=2.22 is FALSE  
FROM (SELECT 1 AS i1, 2 AS i2  
 , 1.0 AS d1, 2.22 AS d2) trs2;  
  
-- = is equal to  
-- < is less than, <= is less than equal to  
-- > is greater than, >= is greater than equal to  
-- != and <> are not equal to  
  
  
-- value BETWEEN begin\_value AND end\_value  
-- compares whether the value is within and/or   
-- equals the begin and end values:  
-- with i1=1 and d2=2.22  
-- 1.5 BETWEEN 1 AND 2.22 is TRUE basically it is  
-- the same as 1<=1.5 AND 1.5<=2.22  
--   
SELECT i1, d2  
 , 1.5 BETWEEN i1 AND d2  
 , i1<=1.5 AND 1.5<=d2 -- same as the between  
FROM (SELECT 1 AS i1, 2 AS i2  
 , 1.0 AS d1, 2.22 AS d2) trs3;  
  
  
  
-- column IN(value1, value2 ... , valueN) will  
-- return 1 (TRUE) is the column value is   
-- matching a value in the list:  
--   
SELECT i1 IN (1,2,3,4) -- TRUE 1  
 , i2 IN (5,6,7,8) -- FALSE 0  
 , d1 IN (1,2,3,4) -- TRUE 1  
 , d2 IN (i1,i2,d1) -- FALSE 0  
FROM (SELECT 1 AS i1, 2 AS i2  
 , 1.0 AS d1, 2.22 AS d2) trs4;  
  
  
  
-- also, we can do some mathematics with numbers  
--   
-- add subtract times divide modulus  
SELECT i1+i2, i1-i2, i1\*i2, i1/i2, i1%i2  
 , d1+d2, d1-d2, d1\*d2, d1/d2, d1%d2  
 , i1+d1, i1-d1, i1\*d1, i1/d1, i1%d1  
FROM (SELECT 1 AS i1, 2 AS i2  
 , 1.0 AS d1, 2.22 AS d2) trs5;

-- so we can add(+) subtract(-) multiply(\*)  
-- divide(/) and modulus(%, returns remainder)

* **Strings** (CHAR, VARCHAR, TEXT)
* most common fields within a database
* contains any UTF-8 friendly keyboard characters
* surrounded by single quotes 'stringvalue'
* **VARCHAR(**length) is for variable widths
* columns with varying or changing widths, such as full\_name column
* String Function **LENGTH(**full\_name) function will vary
* maximum length for the utf8mb4 charset, would be VARCHAR(191), as indexes for MySQL are limited to 768 bytes, 768/4 is 192…so 191 is safest.
* **CHAR**(length**)** is fixed width, so storage allocates that length specified, regardless of what data is inside
* If you specify CHAR(10), the field will have 10 characters of length allocated, so 'Hello' is stored as 'Hello ' (with 5 trailing spaces)
* In MySQL, upon retrieval (SELECT) the trailing spaces are not present, unlike other database servers, though are still there.
* would be used for strings that have the same width for its data rows
* examples could include, postal codes 'R0H 0H0', which is a 7 character Canadian postal code or social insurance number (SIN) at 11 characters '642 643 644'
* these should be used sparingly, and only for smaller constant sized fields
* **TEXT** type is used for content, like a news article, or additional/historical information, or change log data.

-- simple strings, begins/ends with space, noting  
-- data is usually trimmed and not stored this way  
SELECT ' Hi ' AS s1  
 , ' Bye ' AS s2;  
  
-- taking this into a temporary result set, we can perform several string functions  
  
SELECT s1, s2 -- returns:  
 , TRIM(s1), TRIM(s2) -- 'Hi' 'Bye'  
 , LTRIM(s1), LTRIM(s2) -- 'Hi ' 'Bye '  
 , RTRIM(s1), RTRIM(s2) -- ' Hi' ' Bye'  
 , CONCAT(s1,'and',s2) -- ' Hi and Bye '  
 , LENGTH(TRIM(s1)), LENGTH('HĬ') -- LENGTH in bytes  
 , CHAR\_LENGTH(TRIM(s1)), CHAR\_LENGTH('HĬ') -- visual LENGTH  
FROM (SELECT ' Hi ' AS s1  
 , ' Bye ' AS s2) tsr6;  
  
-- usually, strings when stored in the database,  
-- the values are TRIM'd, no need in storing  
-- spaces, a CONCAT usually needs to account  
-- for spaces  
  
SELECT s1, s2 -- returns:  
 , CONCAT(s1,' and ',s2) -- 'Hi and Bye'  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr7;  
  
-- string comparisons and 'H' or 'i' patterns  
SELECT s1, s2 -- returns:  
 , s1=s2, s1<>s2 -- 0 (FALSE) | 1 (TRUE)  
 , s1='Hi' -- 1  
 , s1<>'Bye' -- 1   
 , s1 LIKE 'H%' -- 1, % is 0 to many chars  
 , s1 LIKE 'H\_' -- 1, \_ means must have 1  
 , 'H' LIKE 'H%' -- 1  
 , 'Hi' LIKE 'H%' -- 1  
 , ' Hi' LIKE 'H%' -- 0 if pattern ' H%' then 1  
 , 'H' LIKE 'H\_' -- 0, means any 1 char after  
 , s1 LIKE 'H\_\_' -- 0, means any 2 chars after  
 , s1 LIKE '\_i' -- 1, means any 1 char before  
 , s1 LIKE '\_\_i' -- 0, means any 2 chars bef.  
 , s1 LIKE '%i' -- 1, 0 to many chars bef.  
 , s1 LIKE '%%i' -- 1, but pointless, NEVER DO  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr8;  
  
  
  
-- string comparisons and patterns  
SELECT s1, s2 -- returns:  
 , s2='Bye' -- 1  
 , s2!='Hi' -- 1   
 , s2 LIKE 'B%' -- 1, % is 0 to many chars  
 , s2 LIKE 'B\_' -- 0, \_ must have 1 only  
 , 'B' LIKE 'B%' -- 1  
 , 'Bye' LIKE 'B%' -- 1  
 , ' Bye' LIKE 'B%'-- 0 if pattern ' B%' rtns: 1  
 , 'B' LIKE 'B\_' -- 0, means any 1 char after  
 , s2 LIKE 'B\_\_' -- 1, means any 2 chars after  
 , s2 LIKE '\_e' -- 0, means any 1 char before  
 , s2 LIKE '\_\_e' -- 1, means any 2 chars bef.  
 , s2 LIKE 'B\_e' -- 1, means 1 char in middle  
 , s2 LIKE 'B%e' -- 1, 0 to many chars mid  
 , s2 LIKE '\_y\_' -- 1, 1 char beg end, y mid  
 , s2 LIKE '%y%' -- 1, chars beg end, y mid  
 , 'y' LIKE '%y%' -- 1, 0 chars beg end, y mid  
 , 'y' LIKE '\_y\_' -- 0, 1 char beg end, y mid  
 , s2 LIKE '%%e' -- 1, but pointless, NEVER DO  
 , s2 LIKE 'B%%' -- 1, but pointless, NEVER DO  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr9;  
  
  
-- IN and NOT IN comma delimited list  
SELECT s1, s2 -- returns:  
 , s1 IN('Hi','Bye') -- 1  
 , s2 IN('Hi','Bye') -- 1   
 , s1 NOT IN('Hi','Bye') -- 0  
 , s2 NOT IN('Hi','Bye') -- 0   
 , 'Hello' IN('Hi','Bye') -- 0  
 , 'Hello' IN(s1,s2) -- 0  
 , 'Hi' IN(s1,s2) -- 1  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr10;  
  
  
  
-- NULL checks, null does not mean empty string ''  
SELECT s1, s2, s3 -- returns:  
 , s1=s3 -- NULL  
 , s2<>s3 -- NULL  
 , s1=IFNULL(s3,'') -- 0 - workaround  
 , s2<>IFNULL(s3,'') -- 1 - workaround  
 , s3 IS NULL -- 1  
 , s1 IS NOT NULL -- 1  
 , NULL IN (s1,s2,s3) -- NULL  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2  
 , NULL AS s3) tsr11;

-- AND operator, both conditionals must be TRUE  
SELECT s1, s2 -- returns:  
 , s1='Hi' AND s2='Bye' -- 1 TRUE+TRUE=TRUE  
 , s1='Hi' AND s2='Hi' -- 0 TRUE+FALSE=FALSE  
 , s1='Bye' AND s2='Bye' -- 0 FALSE+TRUE=FALSE  
 , s1='Bye' AND s2='Hi' -- 0 FALSE+FALSE=FALSE  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr12;

-- OR operator, either conditional must be TRUE  
--   
SELECT s1, s2 -- returns:  
 , s1='Hi' OR s2='Bye' -- 1 TRUE or TRUE=TRUE  
 , s1='Hi' OR s2='Hi' -- 1 TRUE or FALSE=TRUE  
 , s1='Bye' OR s2='Bye' -- 1 FALSE or TRUE=TRUE  
 , s1='Bye' OR s2='Hi' -- 0 FALSE or FALSE=FALSE  
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr13;

Typically, most of the comparison items that have been demonstrated are handled in a **WHERE** clause. So if the values of the conditionals are FALSE, the ROW will not display:

-- row will display  
SELECT s1, s2   
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr14  
WHERE s1='Hi';  
  
-- row will display  
SELECT s1, s2   
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr15  
WHERE s1='Hi' AND s2='Bye';  
  
-- row will not display  
SELECT s1, s2   
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr16  
WHERE s2='Hi' OR s1='Bye';  
  
-- row will display   
SELECT s1, s2   
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr17  
WHERE s1 IN ('Hi','Bye') AND s2 LIKE 'B%';  
  
-- row will display, noting that this is the same  
-- as the previous query, as an IN is like doing an  
-- OR comparison, since there is a second predicate  
-- with the LIKE, we must put the OR checks in   
-- brackets so it evaluates it like an IN and   
-- doesn’t change the operations of the query, to  
-- look for s1='Hi' then OR then   
-- the query looking for s1='Bye' AND s2 LIKE 'B%'  
-- as the next filter  
SELECT s1, s2   
FROM (SELECT 'Hi' AS s1  
 , 'Bye' AS s2) tsr18  
WHERE (s1='Hi' OR s1='Bye') AND s2 LIKE 'B%';

* **DATE** (DATE, DATETIME, TIME, TIMESTAMP)
* formats are fixed width and are compared like strings (so must be within single quotes):
* **DATE**: **'YYYY-MM-DD**' (zero padded, 10 character date)
* **TIME**: **'HH:MM:SS**' (zero padded, 8 char, 24 hour time)
* **DATETIME**: '**YYYY-MM-DD HH:MM:SS**' (19 characters)

-- row will display, not that the single digit:  
-- month, day of month, hour, minute, second all  
-- are padded by a zero and databases supports the  
-- this constant length (19 characters for DATETIME  
-- field) character format  
  
SELECT d1, d2  
 , d1<=d2 -- 1   
 , d1=d2 -- 0   
 , d1>d2 -- 0  
 , d1='2020-01-01 00:00:00' -- 1  
 , d2='2020-01-01 00:00:00' -- 0  
 , d1 BETWEEN '2019-01-01 00:00:00'  
 AND '2022-01-01 00:00:00' -- 1  
 , d1 NOT BETWEEN '2019-01-01 00:00:00'  
 AND '2022-01-01 00:00:00' -- 0  
FROM (SELECT '2020-01-01 00:00:00' AS d1  
 , '2021-12-31 23:59:59' AS d2) tsr20;  
  
  
-- NOW function with date numeric functions  
SELECT d1  
 , QUARTER(d1) -- Date Quarter   
 , YEAR(d1) -- Date Year0   
 , MONTH(d1) -- Date Month  
 , DAY(d1) -- Date Day   
 , HOUR(d1) -- Date Hour   
 , MINUTE(d1) -- Date Minute   
 , SECOND(d1) -- Date Second   
 , WEEKDAY(d1) -- 0 Mon thru 7 Sun  
FROM (SELECT NOW() AS d1) tsr21;

-- NOW and common string format functions (  
-- <https://www.w3schools.com/sql/func_mysql_date_format.asp>   
SELECT d1  
 , MONTHNAME(d1) -- Date Month  
 , DAYNAME(d1) -- Date Day   
 , DATE\_FORMAT(d1,'%a, %D of %b %Y %l:%i %p')  
 -- Date String  
FROM (SELECT NOW() AS d1) tsr22;  
  
-- MORE FUNCTIONS:  
-- <https://www.w3schools.com/sql/sql_ref_mysql.asp>

## RRC Polytech Logo