LO7 – SELECT Statement

Select is used to query the database. The result set is simply another table (dynamically created from our database).

* SELECT – Simplest form syntax:
  + SELECT \* FROM Table;
* Specifying columns
  + SELECT expression [,expression, …] FROM Table
  + Expression can be a column, constant, function, or a calculation
  + Can specify all columns, only some columns, and can put columns in a different order than when the table was create
  + Can also give the expression an alias with AS
* Full Syntax:
  + SELECT [DISTINCT | ALL] expression\_list

FROM Tables involved(joins if need be)

[WHERE clause]

[GROUP BY groupByList [HAVING groupSearchCondition] ]

[ORDER BY orderList]

* Operators in SQL
  + MATH operators: \*,/,-,+
  + String concat: ||
  + Relational operators: <,> <=,>=,<>(not equal), =
    - != also works, but not standard
    - Relational operators can be applied to character string (city > ‘London’), but how the evaluation is done based on the ASCII table. Oracle comapres string position by position, until it finds a difference, then it compare the ASCII value of each to determine which is greater. Note that all upper case letters appears in a different location that all lowercase letters in ASCII, so normally compare them as all UpperCase or all LowerCase
  + Boolean Operators: AND, OR, NOT
  + Range Operator
    - Column BETWEEN low AND high
    - Is inclusive, includes low and high values as valid
    - Equivalent to column>= low AND column <= high
  + Comparison Operators:
    - Column IN (constant [,constant,…])
    - Column NOT IN (constant [, constant, …])
    - Column LIKE pattern
      * Wild card characters
        + % - any 0 or more characters
        + \_ - any 1 character
  + Nulls
    - Cannot test for NULL values using equality/inequality
    - Column IS NULL
    - Column IS NOT NULL
* Sorting the results
  + Uses ORDER BY clause
  + Only affects the display order of the results of the query
  + Can order on multiple columns
  + NULLS appear first if ascending or last if descending order
  + Default is ascending order (ASC) and descending order is (DESC)

# Single Row functions

Functions below were not covered in power point document SQL functions

Conversion Functions:

* TO\_CHAR – converts a date/time value to a character string in a specific format. Can also convert a number to a string
  + TO\_CHAR(date, format) or TO\_CHAR(number)
* TO\_DATE – Converts a string in a specific format to an Oracle date type
  + TO\_DATE(value,Format)
* Date and time input mask elements:
  + Dd numeric date
  + Mon – three leter month
  + Month/Month - full name month
  + Mm – numeric month
  + Yy – 2 digit year
  + Yyyy – 4 digit year
  + Hh – hourse, hh24 – 25 hour clock. Mi – minutres, ss – seconds, am – am/pm
* TO\_NUMBER – Converts a string to a number, if compatible

Functions for dealing with NULLS

* Often we need to deal with null values by replacing them with meaningful value in a result set. Ex: Replacing null numbers with 0
* NVL(val1,val2)
  + Oracle Specific – replaces val1 with val2 if val 1 is null
  + Evaluates val1, if it is null return val2. If val1 not null, returns val1
* COALESCE (val1,Val2 [, val3, val4, …])
  + Returns the first non-null value
  + ANSI standard version

Misc Functions

* SYSDATE – returns the current system date/time
* USER – returns the currently logged in user

# Group or Aggregate Functions

* Aggregate functions return a single result row based on groups of rows, rather than on single rows.
* The functions apply to a group of rows, and you get a single value
* Mixing aggregate results and single row results does not work in most implementations of SQL (including oracle).
  + Cant group on ID when grouping on category, each line in Category A would have a different ID
  + If this is desired, Usually would require the use of a reporting tool that would show individuals and summary sub totals together
* Aggregate functions take an expression as a parameter. This is usually a column name, but can also be a constant or calculated value (any valid expression)
* DISTINCT can be passed in as part of the parameter, and this means only unique values in the field would be included in the aggregate function. (Usually used for count to count the unique occurrences of values.)
* Aggregates cannot be used in the WHERE clause, instead they arte put in the HAVING clause
* Aggregates and Nulls:
  + If it encounters a null, it ignores it
  + If there are any nulls, they will be ignored by the aggregate function. So if you want to include them you need to either use NVL or, in the case of COUNT, use \*.

* The functions are:
  + SUM
    - Totals the values in the expression. Ignores nulls
  + AVG
    - Averages the values in the expression, ignores nulls
  + COUNT(expression)
    - Counts the number of non-null values in the expression, including duplicates
  + COUNT(\*)
    - Counts the number of rows in the group ( nulls are included)
  + COUNT(DISTINCT Expression)
    - Counts only unique values, ignores nulls
  + COUNT(DISTINCT NVL(expression, nullReplacement))
    - Counts only unique values, use some replacement for null, which will be counted.
  + Max/Min
    - Find the highest(or lowest) value in the expression (ignores nulls)
* GROUP BY and HAVING Clauses
  + Specify the GROUP BY clause if you want Oracle to group the selected rows based on the value of the expressions – each group will have a single summary row in the results
  + GROUP BY Syntax:
    - Group BY expression1 [,expression 2,…] [HAVING conditions]
    - When expressions are in the GROUP BY, they may now be displayed in the select list
    - Multiple expressions can be grouped on, groups created based on unique value combinations of the expression values.
    - NULLS are treated as their own group if the expression has any
  + HAVING Clause is used to restrict groups that do not meet an aggregate function based condition
    - EX: Having (groups that have a sum > 200)
    - HAVING SUM(amount) > 200;
    - WHERE Clause and HAVING clause each have different jobs. Only use the HAVING Clause for aggregate function conditions and the WHERE Clause for other conditions. This is more efficient.

# Joining Tables

* With normalization, we break related information into multiple tables. We use joins to put the tables back together.

General rules to get meaningful results with joins:

* The columns must have related meaning. For ex:, you could join age to qtyOrdered, but the results would have no meaning
* Normally join on the primary and foreign keys
* All tables must be listed in order for the sustem to perform the join, each table joined successively must have a matching table in the group to join to.

Join Syntax:

SELECT expressionList

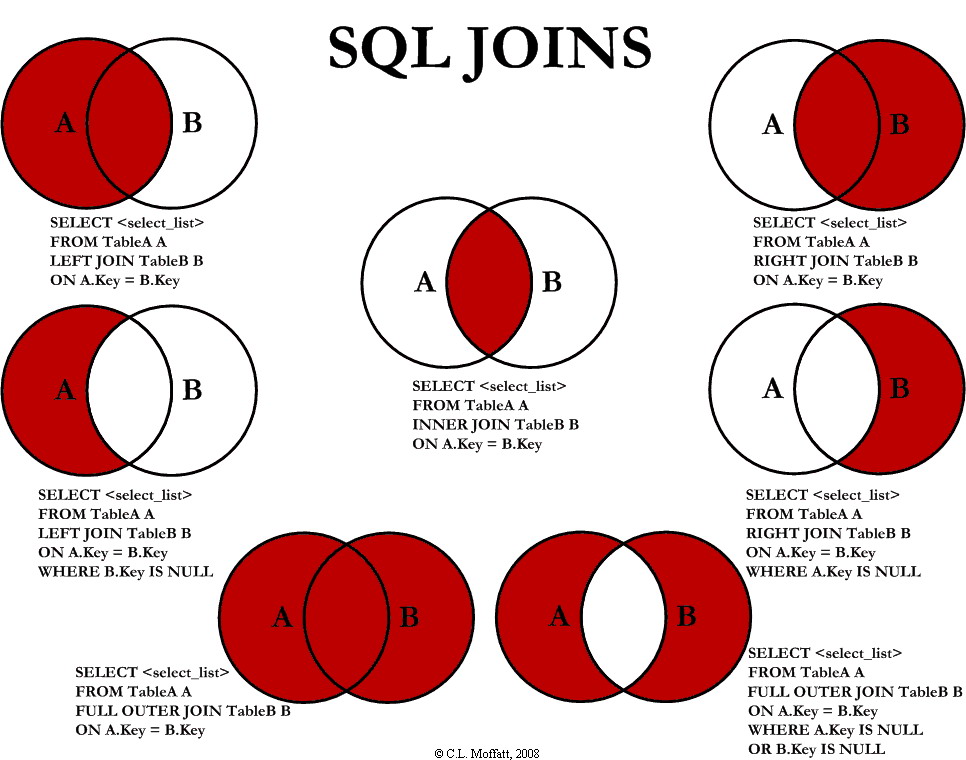
FROM Table1 JOIN table2 ON condition ( table1.fk = table2.pk) [JOIN table3 ON condition ….]

Join Operator

* Usually equality(matches0, but can be any relational operator
* ERDs will help you figure out what join conditions are necessary
* Join columns do not need to be displayed in the expression list

Types of joins:

* Ex: Side A is employee’s and side B is their department
* (Most common)Inner Join: only see things that have a match between both(ex: between FK and PK of 2 tables)
* Top left/Right second most common Ex: If an employee had null for deptID, they would be included if left outer join, but not right
  + It would also include a department that doesn’t have any employee matches if right outer join
* outerJoin but not innerjoin, get all the things that don’t match on a specific side
  + Ex: All dept with no employees, or all employees without a dept
* Full outer Join: get everything
* Full Outerjoin not inner join, get everything that has no matches
* FROM CLASS
* CROSS JOIN - A cross product , or a Cartesian join. All the possible combinations of the rows from each table. Normally not desired or intended
* INNER JOIN – Which is the basic and default join. Returns all of the matches between the tables.
* OUTER JOIN
  + DATA can be missing from the results of an inner join ( the non matches). Outer joins allow these to be regained.
  + LEFT JOIN - Includes all the rows from the table on the left(of the JOIN keyword) whether there is a match or not
  + RIGHT JOIN – Opposite of left join, includes right side rows whether matches or not.
  + FULL JOIN – includes everything, from both sides, regardless if there are matches or not
* SELF JOIN – Join a table to itself, the table must appear more than once in the FROM clause. At least one of the tables must be given an alias, so that we can distinguish one instance of the table from another.



# Set Operators – UNION, INTERSECT, and MINUS

Either combine the results of a query, or compare them

Combine or compare the results of 2 or more queries

* UNION – Isn’t a join, but is a way to combine data from 2 sources (quesry result set)
  + SYNTAX: SELECT … UNION[ALL] SELECT … [ORDER BY sorting];
  + UNION performs a DISTINCT on the results, so the duplicates will be removed, this removal can be prevented with the ALL keyword
  + Number of columns between the queries must be the same
  + The data types of corresponding columns between queries must be compatible.
  + Can order the results , but must be done on the last query
  + Aliases on the first query override all subsequent queries
* INTERSECT
  + Displays only the rows in common between the two queries
  + Same general rules as UNION
* MINUS
  + Show the rows that are in the first query, but not in the second.
  + Same general rules as UNION.