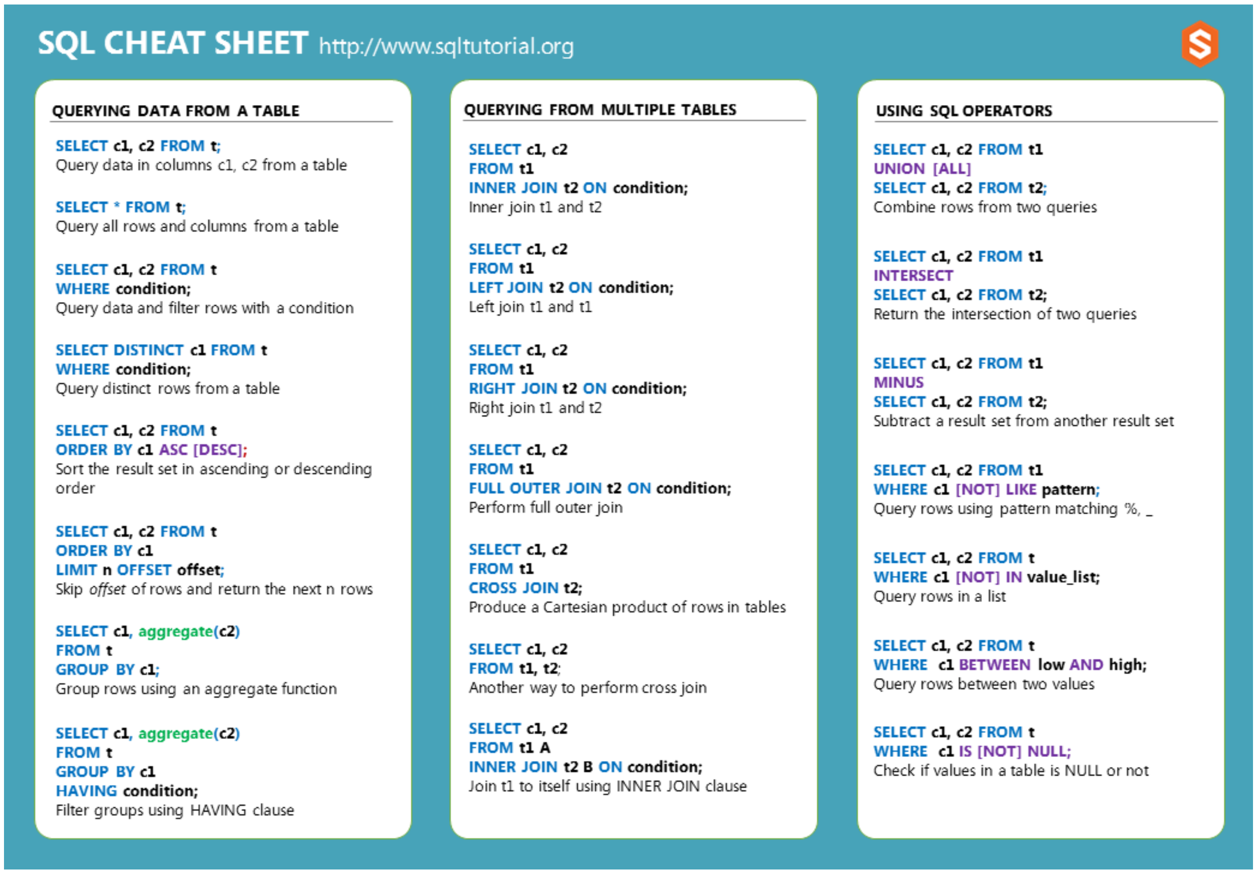
SQL MASTERSHEET

Class	
Created	@Jun 22, 2020 11:03 AM
Materials	
Reviewed	☐
Type	



SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

COMMON COMMANDS:

DML

- **USE** - This allows you to select the objectyou wish to `USE`
- **SELECT**- This allows you to `SELECT` information from within a table you wish to use
    - An asterix * means to select all columns
    - The order of the column_names in the `SELECT` clause is the order its displayed.
- **SELECT DISTINCT -** `SELECT DISTINCT` will only return unique values from the table
- **UPDATE** - If you need to change the contents of the table, use the `UPDATE` statement. Beware of leaving out the where clause, this will update the entire table.
    - E.G `UPDATE director SET director_name = 'Ib' WHERE film_id = 1`
    - This will set the column named `director_name` to Ib where the column `film_id = 1`
- **DELETE** - This allows you to `DELETE` rows from a table. Like update, the where clause must be entered otherwise the entire table will be emptied.

```
DELETE
FROM
    production.product_history
WHERE
    model_year = 2017;
```

- **PRIMARY KEY -** This allows you to set a/multiple column as a primary key when creating a table
    - `PRIMARY KEY(director_id, director_name,''....),`
- **REFERENCES -** This allows to you to set a foreign key to reference a column from another table
    - `FOREIGN KEY(film_id) REFERENCES film_table(film_id)`
    - This is how you set a foreign key in one table that references another table
- **RAND -** Randomise the argument

To create a random decimal number between two values (range), you can use the following formula: A = LL B = UL

`SELECT RAND()*(b-a)+a;`

- **DELETE CASCADE** - A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.

  `ALTER TABLE director`

  `ADD CONSTRAINT film_id`

  `FOREIGN KEY (film_id)`

  `REFERENCES film_table (film_id) ON DELETE CASCADE`

- **WHERE -** The WHERE clause filters the results based on a condition.
- **HAVING -** `HAVING` functions the same as `WHERE` and is to be used where aggregate or `GROUP BY` functions have been used
- **COUNT -** The SQL COUNT() function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. It sets the number of rows or non NULL column values. COUNT() returns 0 if there were no matching rows.
  - Example showing how to `SELECT` from multiple tables
  - `FROM` can efer

  ```
  SELECT(
        SELECT COUNT(*)
      FROM   employees
      ) AS Total_Employees,
      (SELECT COUNT(*)
      FROM   departments
      ) AS No_Of_Departments
  FROM dual
  ```

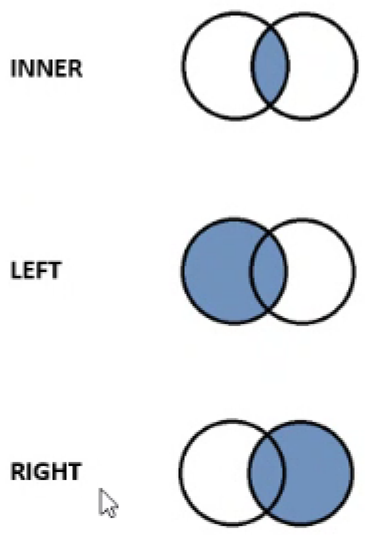
- **AND -** Preceded by a `WHERE` command, this ensures that all criteria is fulfilled
- **OR -** As above but only one of the criteria must be fulfilled
- **OPERATORS:**
  - `<> OR !=` : Not equal to

    - You know the rest!

- **JOIN -** `JOIN` is used to join rows from two different tables.

  Types of Join:

  1. **INNER JOIN** - Matched rows from all tables in query based on the condition being met
  2. **RIGHT JOIN** - All rows from right table with matched rows from left table
  3. **LEFT JOIN** - All rows from left table with matched rows from right table
  4. **FULL JOIN -** Everything is included, even rows that aren't match. A full addition of both tables.



- **Wildcards - CLICK HERE FOR WILDCARD RESOURCE**
- **Date Functions -** CLICK HERE FOR DATE FUNCTIONS

  Common Date Functions

    - **GETDATE() -** To return the current date and time
    - **SYSDATETIME() -** To return the date and time of the computer being used
    - **DATEADD(d/m/y, no. of units, originaldate)** This adds a number of days/months/years to a date specified within the function

        - **DATEDIFF(h/d/m/y, Order Date, Shippeddate)** This shows the difference in a number of days/months/years between two dates. In above example it will be order - shipped.
        - **YEAR(Orderdate)** Will extract the year from a date.
        - **MONTH(Orderdate)** to extract the month from a date.
        - **DAY(Orderdate)** to extract the day from a date
- **Aggregate Functions - CLICK HERE FOR LIST OF FUNCTIONS**
    - Common Aggregate Functions
- **String Functions -** CLICK HERE FOR LIST OF FUNCTIONS

  Common String Funtions:

    - **SUBSTRING**(expression, start, length)
    - **CHARINDEX**('a', 'text') This searches for 'a' in column 'text'
    - **LEFT/RIGHT**(name,5) For the first or last 5 characters
    - **LTRIM/RTRIM** - Used to remove spaces at the beginning/end of string
    - **LEN**(name) - Returns length of the values in name column
    - **REPLACE**(NAME, 'A','1') - To replace all 'A' with '1'
    - **UPPER/LOWER**(NAME) - To convert to all Upper/Lower case
- **BETWEEN -** `BETWEEN` includes values between and as well as the boundary values
- **CONCATENATE -** `CONCAT()` allows the user to combine strings into one field.
- **IS -** `IS` and `IS NOT` be used as an equals sign for null

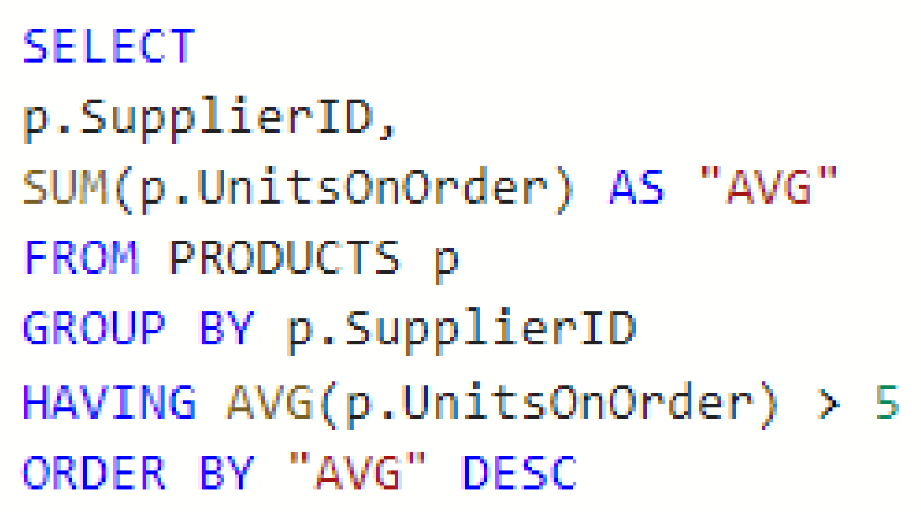
  `SELECT CompanyName`

  `FROM CUSTOMERS`

  `WHERE REGION IS NULL`
- **ROUND -** This is used to limit the amount of decimals of an input number
    - I.E `ROUND(5.6787, 2)` WILL RETURN 5.68
- **ORDER -** Can be ordered by ASC or DESC

SQL SELECT STATEMENT - PROCESSING SEQUENCE

1. **SELECT**
2. **DISTINCT**
3. **FROM**
4. **WHERE**
5. **GROUP BY**
6. **HAVING**
7. **ORDER BY**



```
SELECT
p.SupplierID,
SUM(p.UnitsOnOrder) AS "AVG"
FROM PRODUCTS p
GROUP BY p.SupplierID
HAVING AVG(p.UnitsOnOrder) > 5
ORDER BY "AVG" DESC
```

DDL

- **CREATE** - This allows you to `CREATE` an object
- **ALTER** - This allows you to `ALTER` the database you wish to use
- **DROP** - This deletes a table
- **TRUNCATE** -
- **NOT NULL -** This forces an entry to not be empty, this will cause an error if this condition is not met. E.G

`ALTER COLUMN film_name VARCHAR (10) NOT NULL`

DATA TYPES:

- **VARCHAR (x)** - ***"Variable amount of Chars"*** - Lets SQL know what the type of data will be. x refers to the max number of characters allowed in the entry.
- **CHAR (x)** **-** Data must be at the specified (x) length. If the input is less than the fixed, then blank chars will fill the remaining space. This is good when you know for sure the length of input.

  > **For this reason VARCHAR is memory efficient. However, Char is 50% faster.**

- **INT -** Holds a whole number, positive or negative.
- **DATE/TIME/DATETIME -** Stores Date, Time or both Date and Time
- **DECIMAL (x, y) -** Numbers with a fixed precision and scale. 345.6789 = DECIMAL (7, 4)
- **BINARY-** Used to store binary data such as an image or file
- **FLOAT-** Scientific use (Very large numbers)
- **BIT-** Equivalent to binary (0, 1 or null) *(TRUE FALSE OR UNDEFINED?)*
- **TEXT(size) -** Holds a string with a maximum length of 65,535 bytes

  VARCHAR Vs TEXT

- **TINYTEXT -** Holds a string with a maximum length of 255 characters
- **MEDIUMTEXT -** Holds a string with a maximum length of 16,777,215 characters
- **LONGTEXT -** Holds a string with a maximum length of 4,294,967,295 characters
- **ENUM(val1, val2, val3, ...) -**

Definitions

1. **Primary Key -** A primary key, also called a primary keyword, is a key in a relational database that is unique for each record. It is a unique identifier, such as a driver license number, telephone number (including area code), or vehicle identification number (VIN). A relational database must always have one and only one primary key.
2. **Foreign Keys** - Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

   Foreign Key
3.
4. **One to One relationship** - Where there is one row in one entity can be linked to only one row in another entity.
5. **One to Many Relationship** - In a relational database, a one-to-many relationship exists when one row in table A may be linked with many rows in table B, but one row in table B is linked to only one row in table A. (EXAMPLE: One customer can have many purchases)
6. **Many to Many Relationship -** A many-to-many relationship refers to a relationship between tables in a database when a parent row in one table contains several child rows in the second table, and vice versa. Many-to-many relationships are often tricky to represent. (EXAMPLE: A database used by a school application can be taken as an example. Two of the tables it contains are "Student" and "Subject." In real life, a student will take several subjects simultaneously, while a subject will be studied by several students at a time. This is a many-to-many relationship.)
7. **Junction Table** - When you need to establish a many-to-many relationship between two groups, the simplest solution is to use a Junction Table.

   A Junction Table (sometimes referred to as a "Bridge Table") is a table that contains references to both groups; bridging them together. (Example: Where there is a many to many relationship you can make a junction table with the primary keys of the tables, then form a junction table. This table then forms a **composite primary key.**

8. **Candidate Primary key** - If there are multiple choices for being a primary key, then all the options are called candidates for being the primary key
9. **DML (Data Manipulation Language) -** A data manipulation language (DML) is a computer programming language used for adding (inserting), deleting, and modifying (updating) data in a database. A DML is often a sublanguage of a broader database language such as SQL, with the DML comprising some of the operators in the language.
10. **DDL (Data Definition Language) -** Data Definition Language (DDL) is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

    IMPORTANT NOTE: DROP IS NOT THE SAME AS DELETE - DO NOT CONFUSE
11. **NULL - Null is not nothing, its not 0 or an empty string it is null. A value can be null, but null cannot = null because null is undefined**
12. **First Normal Form**

    A database is in first normal form if it satisfies the following conditions:

    - Contains only atomic values
    - There are no repeating groups

      First Normal Form
13. **Second Normal Form -** A database is in second form if all non-key attributes are fully functional dependent on the primary key. It cannot depend on just part of the primary key.

    Second Normal Form
14. **Third Normal Form -** A database is in third form if in addition to 2nd there is no transitive functional dependency

    By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is *functionally dependent on C. In this case, C is transitively dependent on A via B.*

Third Normal Form

15. **Constraints -** SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. Constraints can be column level or table level.

    ### Common Constraints

    - **NOT NULL** - Ensures that a column cannot have a NULL value
    - **UNIQUE** - Ensures that all values in a column are different
    - **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
    - **FOREIGN KEY** - Uniquely identifies a row/record in another table
    - **CHECK** - Ensures that all values in a column satisfies a specific condition
    - **DEFAULT** - Sets a default value for a column when no value is specified
    - **INDEX** - Used to create and retrieve data from the database very quickly

Rule of Breaking Down Tables

If there is duplicated data, that is a sign that tables can be broken down further.

Best Practise

1. Ensure tables are labelled so people have an idea of the content before reading the table.
2. Duplicated data is to be avoided.
3. Primary keys are part of the design and should be chosen so that it wont change in the future. Must be permanent and static.

4. When you can find a primary key, look for two columns that can form a composite primary key.
5. When creating a database, add a **DROP** statement. E.G
6. Single quotes for value names, double quotes for column names

Questions

1. Does a composite primary key have to be put into a junction table?
    - I think problably no.
2. When there is a defualt value, and you wnt the an item in a row to be default, but other rows to have non-default values, how do you input this?
    - I think you leave it blank, not empty string just blank