**Topics to revise**

* row and page compression mechanism
* cdc
* data compression
* backup compression
* resource governor
* performance data collector
* transparent data encryption
* log shipping and implemented replication as high availability techniques
* database mirroring
* disaster recovery solution
* ssis & ssrs for BI requirements
* cte
* window function
* memory optimized tables
* For mysql on the cmd, statements should be terminated with a semi-colon. This is optional for phpMyAdmin interface
* White space and multiple lines are ignored in the phpMyAdmin interface
* When sorting data in either asc or desc, you can use the ORDER BY with even a column name which is not included in the data being selected
* When filtering a range of data, eg. SELECT \* FROM room WHERE room\_id BETWEEN 5 and 10. NB: the numbers used with the BETWEEN are also included in the results, ie. 5 and 10
* When you combine AND with OR make sure to wrap the conditions in bracket. Since mysql reads from left to right, leaving them without the grouping would cause mysql to execute the statements as it reads through
* Replace multiple OR with IN or NOT IN statement.
  + SELECT \* FROM room WHERE hall\_id = 1 OR hall\_id = 3 OR hall\_id = 4
  + SELECT \* FROM room WHERE hall\_id IN (1,3,4)
  + SELECT \* FROM room WHERE hall\_id NOT IN (1,3,4)

**Pattern search in sql**

1. **The like operators (%, \_, [], ^, - )**

* **NB: the first 2 (% and \_) are used in both sql and mysql. The remaining 3 ([], ^, -) are used in SQL only and are invalid for mysql**
  + % means any number of characters
  + \_ means only one character. The number of \_ used represents the number of characters you want.
  + Either % or \_ can be placed anywhere in the search pattern. Eg.
    - %ent means any number of characters should appear before the word ent
    - \_ent means only 1 character should appear before the word ent
    - \_\_ent means only 2 characters should appear before the word ent
    - ent% means any number of characters should appear after the word ent
    - ent\_ means only 1 character should appear after the word ent
    - ent\_\_ means only 2 characters should appear after the word ent
    - %ent% means any number of characters can appear before and after ent. Ie. the word should simply contain ent
    - \_ent\_ means only 1 character should appear before and after the word ent
    - \_\_ent\_ means only 2 characters should appear before ent while only 1 should appear after ent
    - ent%se means any number of characters should appear between ent and se
    - ent\_se means only 1 character should appear between ent and se
    - ent\_\_se means only 2 characters should appear between ent and se
  + [] means either one of the characters placed inside the bracket. Similar to % and \_, it can be placed anywhere in the search pattern
    - [bdg]ent% means either b, d or g should appear before ent then followed by any number of characters
    - %ent[bdg] means any number of characters should appear before ent, then the last character following ent should be either b, d or g
    - [bdg]e[bdg] means either b,d or g should appear followed by e, then lastly followed by either b, d or g
    - e[bdg]ar means e should first appear followed by either b, d or g then lastly followed by ar
  + ^ or ! is placed inside the [] and does the opposite ie. should not include either of the characters
    - h[oa]t would produce results like with only o or a such as hot, hat: but not hit
    - h[!oat] would produce results like hot, hat: but not hit
    - h[^oa]t would exclude letters with o or a such as hot,hat
  + – is placed inside the [] and means a range. Eg. [a-f] means either of the characters from a to f
* Filtering with the like operators is case insensitive. Eg.
  + SELECT DISTINCT country from new\_sms\_download WHERE country LIKE 'united%'
  + Or
  + SELECT DISTINCT country from new\_sms\_download WHERE country LIKE 'united%'
  + Both produce the same results. And if we have United Kingdom and UNITED STATES in the results, both would be returned

1. **Regular expressions (^ $ . […] [^…] p1|p2|p3 \* + {n} {m,n})**
   * To filter data beginning with some pattern, use ^
   * Eg. SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP '^united'
   * To filter data which ends with some pattern, use $
   * Eg. SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP 'na$'
   * Filter data that contains some text eg. “ana”: SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP 'ana'
   * To filter data which contains some pattern, use ‘’
   * Eg. SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP 'na'
   * To filter data which contains one of a given set of patterns use |
   * Eg. SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP 'ana|uni|ma'
   * To filter data which contains one of a given set of characters use []
   * Eg. SELECT DISTINCT country from new\_sms\_download WHERE country REGEXP '[xyz]'
   * To filter data which contains characters within a range use –
   * Eg. SELECT content FROM new\_sms\_download WHERE content REGEXP '[0-9]'
   * To filter which number of characters should appear before or after a text pattern, use . and ^
   * Eg. SELECT DISTINCT country FROM new\_sms\_download WHERE country REGEXP '^..na'

* Aggregate functions
* An aggregate function is a function which takes an entire column of information and it gives you one (1) answer. Aggregate functions include AVG, SUM, MAX, MIN, COUNT
  + Eg. The average of a column. SELECT AVG(room\_total\_beds) from room
* GROUP BY is often used with aggregate functions to group the avg,sum,max,min or count according to the distinct values of another column. Eg.
  + SELECT country, COUNT(gender) AS population, AVG(id)
  + FROM new\_sms\_download
  + GROUP BY country
* HAVING CLAUSE: NB: the aggregate functions(COUNT,SUM,AVG,MAX,MIN) always return a number as the final result. To give these numbers a condition, ie. maybe you want find where the count >, <, >=, <=. The HAVING clause is only used with the GROUP BY statement as serves as the sql ‘where’ feature for GROUP BY functions. Eg.
  + SELECT country, COUNT(gender) AS population, AVG(id)
  + FROM new\_sms\_download
  + GROUP BY country
  + HAVING population < 100 (usually we’d say WHERE population <100 but since this result is from an aggregate function, we use HAVING)
* Therefore, WHERE statement and HAVING clause perform the same idea of filtering our data. The WHERE statement is used to filter columns which are not the result of aggregate functions. The HAVING clause is used to filter columns which are the result of aggregate functions.
* In short, WHERE is for rows, HAVING is for groups
* Combining the two we can have something like eg.
  + SELECT country, COUNT(gender) AS population, AVG(id)
  + FROM new\_sms\_download
  + WHERE country REGEXP '[a-m]%'
  + GROUP BY country
  + HAVING population < 100;
* NB: for filtering, you use WHERE or HAVING depending on the type of column. However, ORDER BY can be used for normal table columns and columns generated by aggregate functions. Eg.
  + SELECT country, COUNT(gender) AS population, AVG(id)
  + FROM new\_sms\_download
  + WHERE country REGEXP '[a-m]%'
  + GROUP BY country
  + HAVING population < 100
  + ORDER BY population DESC, country ASC
* SubQueries
* First and foremost, sql runs queries from the innermost query onto the outermost. Eg
  + SELECT country FROM countries
  + WHERE population =
  + (

SELECT MIN(population) FROM countries /\*\*run innermost query first\*/

* + )
* Eg.2
  + SELECT country, AVG(id) as AvgID FROM new\_sms\_download
  + GROUP BY country
  + HAVING AvgID >
  + (SELECT AVG(id) FROM new\_sms\_download) /\*\*run innermost query first\*/
  + ORDER BY AvgID ASC, country ASC
* NB: innermost subqueries are not composed of only aggregate functions as in the previous 2 examples. See below
  + SELECT name, phonecode from country
  + WHERE phonecode IN
  + (
  + SELECT phonecode FROM country /\*\*innermost query\*/
  + WHERE phonecode > 500
  + )
  + ORDER BY phonecode ASC, name ASC
* JOINS
* NB 1: the types of JOINS are inner join, left join, right join, full outer join
* NB 2: regardless the kind of join being used, identify which table is being used as the left and which is being used as the right. In the example below
  + SELECT \* FROM country INNER JOIN/LEFT JOIN/RIGHT JOIN/FULL OUTER JOIN state ON country.id = state.country\_id;
  + Always, the left table is the one which we are selecting from ie. country.
  + The right table is the one which is being joined onto the existing table. Here, ie. state
  + Use the practicejoins excel file to observe the key differences in the results produced by each type of join
* Unions
* Combine multiple sql queries into a single table result rather than as separate table results
* **NB:** whenever you use unions, your columns returned in each query has to be the same
* Sometimes the different queries used for the UNION may have the same rows in some. By default UNION removes duplicates and leaves out only 1. If you want all results use **UNION ALL**

* **VIEWS**
* A view is a virtual table whose data is based on the results of an SQL statement. They do not the real hold data but are merely a way to efficiently query your database to prevent waste of time and repetitions. It is a virtual table but it still acts like a table in sql therefore you can perform select statements to view the data in them.
* To create a view, simply write down and test a usual sql statement that you would like to convert into a view. Then optionally enclose the query in brackets preceded by “CREATE VIEW [view name] AS” eg.
* CREATE VIEW myCustomView AS (SELECT \* FROM country)
* **STORED PROCEDURE**
* A stored procedure is a prepared SQL code that you save, so the code can be reused over and over again.
* Similar to views creating a stored procedure simply requires you to write down and test the usual sql statement that you’d like to convert into a stored procedure. Then optionally enclose the query in brackets preceded by “CREATE PROCEDURE [procedure name] AS” eg.

|  |  |
| --- | --- |
| **View** | **Stored procedure** |
| Represents a virtual table | Represents a precompiled database query that can do more than just displaying a table data |
| Takes a query definition and executes it to produce the result. So may take some to execute if the query is a large one | Since the query definition is precompiled, it improves the efficiency and security of the application |
| Can contain only one single SELECT query | Can contain several statements, loops, IF ELSE, etc. |
| Cannot modify a table | Can modify tables |
| Does not take parameters | Can take parameters to be used by the procedure to perform a query |

**SQL SERVER shortcuts to remembering some commands**

* The following can be renamed using sp\_rename
* **sp\_rename** 
  + **table**\_old\_name, table\_new\_name
  + **column**\_old\_name, column\_new\_name ‘COLUMN’
  + **storedProcedure**\_old\_name, storedProcedure\_new\_name
  + **trigger**\_old\_name, trigger\_new\_name

* The following can be retrieved from information\_schema
* **select \* from information\_schema.**
  + **tables** => all tables
  + **columns** => all columns
  + **table\_constraints** => all constraints on columns
  + **views** => all views
  + **routines** => all stored procedures
* The following can be deleted with the drop statement
* **DROP**
  + **Database** database\_name
  + **Table** table\_name
  + **Column** column\_name
  + **View** view\_name
  + **Procedure** procedure\_name
  + **Trigger** trigger\_name
  + **Constraint** constraint\_name
  + **Unique/Foreign Key/Primary Key** constraint\_name

* The following can be edited with the alter statement
* **ALTER**
  + **Database** database\_name
  + **Table** table\_name
  + **Column** column\_name
  + **View** view\_name
  + **Procedure** procedure\_name
  + **Trigger** trigger\_name
* The following can be created with the add statement
* **ADD**
  + Variable\_name
  + **Primary Key**(col\_name)
  + **Foreign Key**(col\_name)
  + **Unique**(col\_name)