DATA FOLKZ

CATAPULT DATA LEADERS

STRUCTURED QUERY LANGUAGE

DATA FOLKZ
CATAPULT DATA LEADERS

RDBMS

Understanding relational databases:

Relational databases store data in tables as rows. They are based on a branch of algebraic set theory known as relational algebra.

Examples of relational databases:

MySQL, PostgreSQL, SQLlite, Ingres, Oracle, Apache Derby, Maria DB, Volt DB, MemSQL, MSAccess

Some terminology:

Entity: Something of interest to the database user community. Example: Customer, location, account etc.

Column: An individual piece of data stored in a table.

Row: A set of columns that completely describe an entity. Also called a record.

Table: A set of rows held either in memory(Non persistent storage) or on disk (persistent storage)

Result set: Another name for non persistent table, generally the result of an SQL query.

Primary key: One or more columns that can be used as a unique identifier for each row in a table.

Foreign key: one or more columns that can be used together to identify a single row in another table.

MySQL data types:

Character Data:

Character data can be stored as either fixed-length or variable-length strings; the difference is that fixed-length strings are right-padded with spaces and always consume the same number of bytes, and variable-length strings are not right-padded with spaces and don't always consume the same number of bytes

For Example:

Char (20) //fixed length

Varchar (20) //Variable length

The maximum length for char columns is currently 255 bytes, whereas varchar columns can be up to 65,535 bytes

Oracle Database is an exception when it comes to the use of varchar.

Oracle users should use the varchar2 type when defining variable-length

character columns.

Text data

If you need to store data that might exceed the 64 KB limit for varchar columns, you will need to use one of the text types.

Text type	Maximum number of bytes
Tinytext	255
Text	65,535
Mediumtext	16,777,215
Longtext	4,294,967,295

Numeric data:

MySQL has several different numeric data types. The most commonly used numeric types are those used to store whole numbers.

When specifying one of these types, you may also specify that the data is *unsigned*, which tells the server that all data stored in the column will be greater than or equal to zero.

Type	Signed range	Unsigned range
Tinyint	**−128 to 127**	**0 to 255**
Smallint	**−32,768 to 32,767**	**0 to 65,535**
Mediumint	**−8,388,608 to 8,388,607**	**0 to 16,777,215**
Int	**−2,147,483,648 to 2,147,483,647**	**0 to 4,294,967,295**
Bigint	**−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807**	**0 to 18,446,744,073,709,551,615**

For floating-point numbers (such as 3.1415927), you may choose from the numeric

types shown in Table 2-4.

Table 2-4. MySQL floating-point types

Type	Numeric range
Float(*p*,*s*)	−3.402823466E+38 to −1.175494351E-38 and 1.175494351E-38 to 3.402823466E+38
Double(*p*,*s*)	−1.7976931348623157E+308 to −2.2250738585072014E-308 and 2.2250738585072014E-308 to 1.7976931348623157E+308

Temporal data:

Along with strings and numbers, you will almost certainly be working with information about dates and/or times. This type of data is referred to as *temporal data.*

***Example**: The date when a customer's order was shipped, Employee's birth date etc.*

MySQL includes data types to handle all of these situations. Table 2-5 shows the temporal data types supported by MySQL.

MySQL temporal types

Type	Default format	Allowable values
Date	YYYY-MM-DD	1000-01-01 to 9999-12-31
Datetime	YYYY-MM-DD HH:MI:SS	1000-01-01 00:00:00 to 9999-12-31 23:59:59
Timestamp	YYYY-MM-DD HH:MI:SS	1970-01-01 00:00:00 to 2037-12-31 23:59:59

DDL and DML statements:

DDL(Data Definition Language) statements or commands are used to define and modify the database structure of your tables or schema. When you execute a DDL statement, it takes effect immediately.

Some DDL commands are:

- CREATE – to create table (objects) in the database
- ALTER – alters the structure of the database
- DROP – delete table from the database
- TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed
- COMMENT – add comments to the data dictionary
- RENAME – rename a table

CREATE :

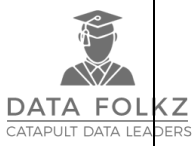
The create table statement (query) to create a table is given below:

CREATE TABLE <table name> (
<attribute name 1><data type 1>,
...
<attribute name n><data type n>);

Example:

CREATE TABLE STUDENT (StudID NUMBER, Name VARCHAR);

ALTER :



The alter table statement to make modifications to the table structure such as Key constraints, Column size, etc.

ALTER TABLE <table name> ADD CONSTRAINT <constraint name> PRIMARY KEY(<attribute list>);

Example:

ALTER TABLE STUDENT ADD CONSTRAINT NOT NULL PRIMARY KEY (StudID);

```
ALTER TABLE table_name
ADD new_column_namecolumn_definition
[ FIRST | AFTER column_name ];
```

Modify column in table

Syntax

The syntax to modify a column in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
MODIFY column_namecolumn_definition
[ FIRST | AFTER column_name ];
```

Drop column in table

Syntax

The syntax to drop a column in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Rename column in table

Syntax

The syntax to rename a column in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
CHANGE COLUMN old_namenew_name
column_definition
[ FIRST | AFTER column_name ]
```

DROP :

The drop table statement (query) to delete a table is given below:

DROP TABLE <table name>;

Example:

DROP TABLE STUDENT;

RENAME

RENAME TABLE
tbl_name TO *new_tbl_name*
[, *tbl_name2* TO *new_tbl_name2*] ...

For example, to rename a table named old_table to tonew_table, use this statement:

RENAME TABLE old_table TO new_table;

TRUNCATE

TRUNCATE[TABLE]*tbl_name*

`TRUNCATE TABLE` empties a table completely.

DML Statements:

Data Manipulation Language (DML) statements or commands are used for managing data within tables. Some commands of DML are:

Some commands of DML are:

- **SELECT** – retrieve data from a database
- **INSERT** – insert data into a table
- **UPDATE** – updates existing data within a table
- **DELETE** – deletes all records from a table, the space for the records remain
- **MERGE – UPSERT** operation (insert or update)
- **CALL** – call a PL/SQL or Java subprogram
- **LOCK TABLE** – control concurrency

Insert:

The insert statement is used to add new row to a table.

INSERT INTO <table name> VALUES (<value 1>, ...<value n>);

Example:

INSERT INTO STUDENT VALUES (101,'Ravi');

UPDATE *:*

The update statement is used to change values that are already in a table.

UPDATE <table name> SET <attribute> = <expression> WHERE <condition>;

Example:

UPDATE STUDENT SET Name = 'Raju' WHEREStudID=101;

DELETE:

The delete statement deletes row(s) from a table.

DELETE FROM <table name> WHERE <condition>;

Apart from these statements, some statements are also used to control the transaction made by DML statements. The commands used for this purpose are called Transaction Control (TCL) statements. It allows statements to be grouped together into logical transactions. Some commands of TCL are:

- COMMIT – save work done.
- SAVEPOINT – identify a point in a transaction to which you can later roll back.
- ROLLBACK – restore database to original since the last COMMIT.

Basic SQL queries:

Selecting all records from a table and displaying all columns:

SELECT * FROM Employees;

Selecting all records from a table and displaying a specified column:

SELECT emp_id FROM Employees;

Selecting all records from a table and displaying multiple columns separated by commas:

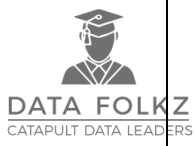
SELECT emp_id, lname FROM Employees;

Displaying data for a given condition:

SELECT emp_id, lname
FROM Employees
WHERE emp_id = '101';

Displaying data for a given condition and sorting the output:

SELECT emp_id, lname
FROM Employees
WHERE city = 'Hyderabad'
ORDER BY emp_id;



Displaying data for a given condition and sorting the output on multiple columns, one column sorted in reverse order:

SELECT emp_id, lname
FROM Employees
WHERE city = 'Hyderabad'
ORDER BY emp_id, lname DESC;

Displaying data for a given condition and sorting the output using an integer in the place of the spelled-out column name:

SELECT emp_id, lname
FROM Employees
WHERE city = 'Hyderabad'
ORDER BY 1;

Displaying data for a given condition and sorting the output by multiple columns using integers, the order of the columns in the sort is different than their corresponding order after the SELECT keyword:

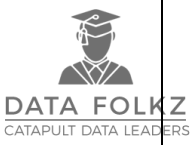
SELECT emp_id, lname
FROM Employees
WHERE city = 'Hyderabad'
ORDER BY 2, 1;

GROUP BY

"GROUP BY" is similar to "ORDER BY," but it will aggregate data that has similarities. For example, if you have any duplicates in your data, you can use "GROUP BY" to count the number of duplicates in your fields.

Here is your SQL query:

SELECT fname, lname
FROM Employees
WHERE city = "Hyderabad"
GROUP BY lname;



Counting the Records in a Table

A query can be issued on a table to get a quick count of the number of records in the table or on the number of values for a column in the table. A count is accomplished by the function COUNT.

The syntax of the COUNT function is as follows:

SELECT COUNT(*) FROM Employees;

Counting the number of values for **emp_id** in the **Employees** table:

Column Aliases

Column aliases are used to rename a table's columns for the purpose of a particular query.

SELECT COLUMN_NAME ALIAS_NAME
FROM TABLE_NAME;

AND

AND allows you to add additional criteria to your WHERE statement.

SELECT fname, lname FROM Employees

WHERE city = 'HYDERABAD' AND joinDate BETWEEN '2010-01-01' AND '2017-12-31';

Regular expressions and wild cards in SQL:

- A wildcard character is used to alternate for any other character or characters in a string.
- To **search for data** in a table, SQL wildcards are used.
- One or more wildcards can be **combined**.
- **Wildcards** do the same function as "**regular expressions**".

SQL Wild cards:

% - An alternate for more than 0 characters.

_ - An alternate for a single character.

[charlist] – Specifies range of characters and sets to match.

[^charlist] and [!charlist] – Specifies ranges of characters and sets not to match.

Examples:

SELECT * FROM Employees WHERE City LIKE 'H%';

This example selects all developers with a City starting with any character, followed by "ondon":

SELECT * FROM Developers

WHERE City LIKE '_ondon';

SELECT * FROM Developers

WHERE City LIKE 'D_l_i';

SQL is excellent at aggregating data:

- COUNT counts how many rows are in a particular column.
- SUM adds together all the values in a particular column.
- MIN and MAX return the lowest and highest values in a particular column, respectively.
- AVG calculates the average of a group of selected values.

Examples:

SELECT min(salary) FROM Employees;

SELECT max(salary) FROM Employees;

SELECT avg(salary) FROM Employees WHERE city = “Hyderabad”;

SELECT sum(salary) FROM Employees WHERE city = “Chennai”;

Joins in SQL:

Joins are a way of retrieving information from two, three or more related tables. Suppose there are 2 tables. One is Employee table and the other being department table. We may need to get data from both these tables.

Cartesian product:

This is every permutation of the 2 tables.

INNER JOIN

The inner JOIN is used to return rows from both tables that satisfy the given condition.

SELECT e.lname,e.emp_id,d.dept_name

FROM Employees e INNER JOIN department d

ON e.emp_id = d.emp_id;

OUTER JOIN

MySQL Outer JOINs return all records matching from both tables .

It can detect records having no match in joined table. It returns **NULL** values for records of joined table if no match is found.

LEFT JOIN

Assume now you want to get titles of all movies together with names of members who have rented them. It is clear that some movies have not being rented by any one. We can simply use **LEFT JOIN** for the purpose.

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. **Where no matches have been found in the table on the right, NULL is returned.**


```
SELECT A.`title` , B.`first_name` , B.`last_name`

FROM `movies` AS A

LEFT JOIN `members` AS B

ON B.`movie_id` = A.`id`
```

RIGHT JOIN

RIGHT JOIN is obviously the opposite of LEFT JOIN. The RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

In our example, let's assume that you need to get names of members and movies rented by them. Now we have a new member who has not rented any movie yet

```
SELECT  A.`first_name` , A.`last_name`, B.`title`
FROM `members` AS A
RIGHT JOIN `movies` AS B
ON B.`id` = A.`movie_id`
```

Sub queries:

A sub query is a select query that is contained inside another query.

Example:

SELECT salary FROM Employees WHERE salary =(SELECT MIN(salary) from Employees);

Sub-Queries Vs Joins!

When compare with Joins , sub-queries are simple to use and easy to read. They are not as complicated as Joins

But sub-queries have performance issues. Using a join instead of a sub-query can at times give you upto 500 times performance boost.

Given a choice, it is recommended to use a JOIN over a sub query.

Co-related sub queries:

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.

The main difference between a correlated subquery and a simple subquery is that **correlated subqueries reference columns from the outer table**.

Example:

SELECT e.fname,e.salary,e.dept_id

FROM employee e

WHERE e.salary>(SELECT avg(salary)

FROM employee e2

GROUP BY dept_id

HAVING e.dept_id = e2.dept_id);

CASE Clause:

SQL handles if/then logic using CASE clause.

This statement is followed by atleast one pair of WHEN and THEN statements.

Example:

update employee

set gender =

CASE gender

WHEN 'F' THEN 'M'

WHEN 'M' THEN 'F'

END;

Date and Time functions in SQL:

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified

Name	Description
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(),LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods

Name	Description
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)

Name	Description
YEAR()	Return the year
YEARWEEK()	Return the year and week

Example:

Q: Print the current date

A. select curdate(); //prints the current date

Q: If you want the date to be printed in specific format

A. select date_format('2018-03-21','%W %M %Y');

A : select date_format(curdate(),'%W %M %Y');

Q: To print the name of the dat

selectdayname(curdate());

Q: DATE_FORMAT() function

A: SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y') today;

Q: DATE_DIFF()

A. select datediff(curdate(),'2018-03-01');

```
SELECT    orderId,
DATEDIFF(reqDate, shippedDate) daysLeft
FROM     orders
ORDER BY daysLeft DESC;
```

Q: Display current date and time together

A: select now();

Duplicate Rows deletion in MySQL:

Consider the below table users.

Create table users (

Id int primary key,

Name varchar(20),

Email varchar(50));

Insert into users values (1, “Sam”, “sam@gmail.com”);

Insert into users values (2, “Latha”, “Latha@gmail.com”);

Insert into users values (3, “Veda”, “veda@gmail.com”);

Insert into users values (4, “Sam”, “sam@gmail.com”);

Insert into users values (5, “Nitya”, “Nitya@gmail.com”);

Query to return duplicate entries from the user table:

Select email, count(email) as cnt from users

Group by email

Having cnt> 1;

The above query will show you the rows with duplicate users (here users with the same mail id.)

Query to delete duplicate entries from the user table:

```
Delete t1 from users t1
Inner join users t2
Where
t1.id < t2.id
And t1.email = t2.email;
```

THANK YOU