mysql -h cxln2.c.thelab-240901.internal -u sqoopuser -pNHkkP876rp

On the MySQL prompt, run below SQL commands:

# Switch to a database

use retail\_db;

# Create the table, ignore the errors if table is already created

create table t\_test(name varchar(10));

# Delete any existing rows

delete from t\_test;

# Insert new rows

insert into t\_test values("John");

# Select the data

select \* from t\_test;

MySQL Database Terminology

One of the freely available, reliable and popular databases is MySQL and we will be studying more about same in this series.

Before we proceed to explain the MySQL database system, let us have a look at few definitions related to the database which are in general applicable to any other relational databases as well.

1. **Database** - A database is a collection of tables, with related data.
2. **Table** - A table is a matrix with data. A table in a database looks like a simple spreadsheet.
3. **Column** - One column (data element) contains data of one and the same kind, for example, the column postcode.
4. **Row** - A row (= tuple, entry or record) is a group of related data, for example, the data of one subscription.
5. **Redundancy** - Storing data twice in more than one tables, redundantly to make the system faster.
6. **Primary Key** - A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row. A table will have only one Primary Key
7. **Unique Key** - A unique key is a combination of one or more columns which can uniquely identify a row in the table. That combination cannot occur more than once in one table excluding rows having NULL values in key columns. A table can have more than one unique key.
8. **Compound Key** - A compound key (composite key) is a key that consists of multiple columns because one column is not sufficiently unique. Another term for the unique key which has more than one column.
9. **Foreign Key** - A foreign key is a linking pin between two tables.
10. **Index** - An index in a database resembles an index at the back of a book.
11. **Referential Integrity** - Referential Integrity makes sure that a foreign key value always points to an existing row.

# MySQL Database Terminology

One of the freely available, reliable and popular databases is MySQL and we will be studying more about same in this series.

Before we proceed to explain the MySQL database system, let us have a look at few definitions related to the database which are in general applicable to any other relational databases as well.

1. **Database** - A database is a collection of tables, with related data.
2. **Table** - A table is a matrix with data. A table in a database looks like a simple spreadsheet.
3. **Column** - One column (data element) contains data of one and the same kind, for example, the column postcode.
4. **Row** - A row (= tuple, entry or record) is a group of related data, for example, the data of one subscription.
5. **Redundancy** - Storing data twice in more than one tables, redundantly to make the system faster.
6. **Primary Key** - A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row. A table will have only one Primary Key
7. **Unique Key** - A unique key is a combination of one or more columns which can uniquely identify a row in the table. That combination cannot occur more than once in one table excluding rows having NULL values in key columns. A table can have more than one unique key.
8. **Compound Key** - A compound key (composite key) is a key that consists of multiple columns because one column is not sufficiently unique. Another term for the unique key which has more than one column.
9. **Foreign Key** - A foreign key is a linking pin between two tables.
10. **Index** - An index in a database resembles an index at the back of a book.
11. **Referential Integrity** - Referential Integrity makes sure that a foreign key value always points to an existing row.

**Date and Time Types**

* **DATE** - Date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. Ex - 2018-06-28
* **DATETIME** - Date/time in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. Ex - 2018-06-28 18:05:00
* **TIMESTAMP** - Timestamp between midnight, January 1st, 1970 and sometime in 2037. The main difference between DATETIME and TIMESTAMP is that DATETIME is constant while TIMESTAMP is affected by the time\_zone setting.
* **TIME** - Time in an HH:MM:SS format
* **YEAR(M)** - Year in 2 or 4 digit format. M -> 2 -> YEAR between 1970 and 2069. M -> 4 -> YEAR between 1901 and 2155.

**String Types**

* **CHAR(M)** - Fixed-length string between 1 and 255 chars. Right-padded with spaces to the specified length when stored. M is optional, defaults to 1. Ex - CHAR(5)
* **VARCHAR(M)** - variable-length string between 1 and 255 chars. M is must. Ex - VARCHAR(30)
* **BLOB** or **TEXT** - "Binary Large Objects", Max length of 65535 chars. Store binary data like images. BLOB - Sorts and comparisons on the stored data are case sensitive. TEXT - Sorts and comparisons are not case sensitive. No need to specify length.
* **TINYBLOB** or **TINYTEXT** - BLOB or TEXT column with a max length of 255 chars. No need to specify length.
* **MEDIUMBLOB** or **MEDIUMTEXT** - BLOB or TEXT column with a max length of 16777215 chars. No need to specify length.
* **LONGBLOB** or **LONGTEXT** - BLOB or TEXT column with a max length of 4294967295 chars. No need to specify length.
* **ENUM** - List of fixed items from which the value must be selected. Ex - mylist ENUM('0','1','2')

# Creating and Dropping a table

A table is an object which stored data in rows and columns.

The table creation command requires the following details:  
Name of the table Name of the fields Definitions for each field

**Basic Syntax:**

create table table\_name (

column\_name1 column\_type1 [default <values>]

,column\_name1 column\_type1 [default <values>]

);

**Full Syntax:**

<https://dev.mysql.com/doc/refman/8.0/en/create-table.html>

**Example:**

use retail\_db;

create table deptt(

depid int not null,

depname varchar(30) not null default 'New',

depopendate datetime not null default CURRENT\_TIMESTAMP

);

Try creating above table and if succeeds, you will see a message like below one, else you will see relevant error message:  
Query OK, 0 rows affected (0.01 sec)

View the table definition using below commands:

describe deptt;

OR

show create table deptt;

show columns from deptt;

show table status like 'deptt';

Try selecting the data from the table. You would see 0 rows because no data has been inserted yet into the table.

select \* from deptt;

**Dropping a table**  
Tables can be dropped using 'drop table' command. A table can not be dropped if it is being used by someone else or if the table has child tables enforced by Foreign Keys and child table has records.

drop table deptt;

# Inserting data into a table

Data can be inserted into a table in many ways.

Syntax:

insert into table\_name ( field1, field2,...fieldN )

values

( value1, value2,...valueN );

Insert statement example:

insert into deptt (depname,depcity, depstreet,depopendate)

values('Accounts','Bengaluru','Daker Street',NOW());

select last\_insert\_id();

insert into deptt (depname,depcity, depstreet,depopendate)

values('HR','Delhi','Baker Street',NOW());

select last\_insert\_id();

insert into deptt (depname,depcity, depstreet,depopendate)

values('Finance','Mumbai','Laker Street',NOW());

select last\_insert\_id();

A value inside 'values' expression can be a SELECT statement as well.

insert into deptt (depname,depcity, depstreet,depopendate)

VALUES('Finance','Mumbai',(SELECT ... FROM <some\_table>),NOW());

In above SQL, department street is being selected from some other table and is inserted into department table.

Whole 'values' clause can also be replaced with SELECT from a table which is essentially copying data from a table into another.

insert into deptt(depname,depcity, depstreet,depopendate) SELECT depname,depcity, depstreet,depopendate FROM deptt;

Selecting data from a table

Syntax:

select field1, field2,...fieldN

[from table\_name1, table\_name2...]

[where clause]

[limit [offset M ][limit N]]

When selecting a fixed value, FROM clause is not mandatory.

select 'Hello';

* select \* -> Return all fields
* select col1, col2 -> Return specified fields
* offset -> To skip initial records
* limit -> How many rows to return

SELECT examples:

select \* from deptt;

select d.\* from deptt d;

select \* from deptt d;

select depid, depname, depcity from deptt limit 1;

select depid, depname, depcity from deptt limit 1,2; -- skip 1 and return only 2 rows

**Aliases**

Tables and Columns can be given alias names for easy readability.

select e.empfname "First Name" from emp e;

**Selecting unique values**  
Values in columns of a table may repeat across the rows of the table. To get unique values, use select distinct.

select distinct empfname from emp;

Filtering - WHERE clause

To select the rows meeting only certain criteria, use WHERE clause.  
**Syntax:**

SELECT col1,col2 FROM table1 WHERE conditon1 AND/OR condition2 .....

SELECT col1,col2 FROM table1 WHERE (col1='123' and col2='456') OR col3='4';

**Operators:**

* Equal to: A = B
* Not equal to: A != B
* Greater than: A > B
* Less than: A < B
* Greater than or equal to: A >= B
* Less than or equal to: A <= B
* Between 2 values including the values: BETWEEN A and B
* Like (contains): A like %aker%

Example:

select \* from deptt where depcity = 'Mumbai';

select \* from emp where empsal > 2300;

select \* from emp where empsal between 2400 and 2600;

All these operators can be used on the number as well as strings. When needed, numbers are implicitly converted into strings before comparison.

select \* from deptt where depid like '%3%';

select \* from deptt where depid like '%x%';

select \* from deptt where depid = 'hello';

This 'where' clause can be used in insert/update/delete statements too.

# Updating data in a table

Values of one or more columns of a table can be updated with the 'update' command to store new required values.

**Syntax:**

update table\_name SET field1 = new-value1, field2 = new-value2

[where Clause]

Example:

update deptt

set depcity = 'Pune' where depstreet = 'Laker Street';

update deptt

set depopendate = STR\_TO\_DATE('20170610','%Y%m%d'), depstreet = 'Saker Street' where depcity = 'Mumbai' order by depid desc limit 1;

select \* from deptt ;

Deleting data from a table

* One or more rows of a table can be deleted with 'delete' command to get rid of the data which is no more needed.
* Delete operation will reduce the number of rows in the table which may help in faster select/insert/update on the table.

**Syntax:**

delete from table\_name [where clause]

delete from deptt

where depstreet = 'Laker Street'

;

select \* from deptt;

# Sorting the query results

By default, select can give rows in any order.

To get the data in order by certain columns, use 'order by'.

**Syntax:**

select field1, field2,...fieldN table\_name1, table\_name2...

order by field1, [field2...] [asc[desc]]

select \* from deptt order by depcity asc,depstreet desc;

Sorting columns in 'order by' can also be referred to by column position in the 'select' expression rather than actual column names.

In below example, the result will be sorted by 2nd and 5th columns of the output.

select \* from deptt order by 2,5;

# Joins - Selecting from multiple tables

Practically, data from more than one table is required to serve end-user applications.  
For example, a web page might need to show employee name, employeed id, deptt id, deptt city and salary in tabular format.  
In such case, tables having department details and employee details need to be joined to get required output.  
Let's prepare the data and run a join query.

drop table deptt;

create table deptt(

depid int not null auto\_increment,

depname varchar(30) not null,

depcity varchar(20) not null,

depstreet varchar(30) not null,

depopendate date,

primary key ( depid )

);

insert into deptt (depname,depcity, depstreet,depopendate)

values('Accounts','Bengaluru','Daker Street',now());

insert into deptt (depname,depcity, depstreet,depopendate)

values('HR','Delhi','Baker Street',now());

insert into deptt (depname,depcity, depstreet,depopendate)

values('Finance','Mumbai','Laker Street',now());

drop table emp;

create table emp(

empid int not null auto\_increment,

empfname varchar(100) not null,

emplname varchar(100) not null,

depid int not null,

doj date,

empsal int not null,

mgrempid int,

primary key ( empid ),

foreign key (depid) references deptt(depid) on delete restrict

);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Sam','Fox',1,'2016-06-22',2200,null);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('John','Doe',1,'2017-05-17',1600,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Beck','Dot',2,'2014-09-10',2500,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Jack','Den',2,'2017-10-28',2400,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Emil','Bon',3,'2018-01-20',2100,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Siera','Leck',3,'2014-02-17',2000,4);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Koe','Hoker',3,'2016-03-16',2900,4);

So, now we have 3 departments and 7 employees.  
We want to display department and employee details in a single output.  
Here, deptt and emp tables are related to each other via depid column and generally, and typically such a column is used to join 2 tables.

select e.empid, e.empfname, e.emplname, e.depid, e.doj, e.empsal, d.depname, d.depcity

from emp e, deptt d

where e.depid = d.depid

order by empid,empfname,emplname

;

# Self-Joins

Self-Join means joining of a table with itself.

In above 'emp' table, column 'mgrempid' stores employee id of manager of each employee and manager is also an employee existing in same 'emp' table.

So, if we want to display employee name followed by employee's manager name in same rows, then self-join has to be used.

select e.empid, concat(e.empfname,' ',e.emplname) "employee", e.mgrempid, concat(m.empfname,' ',m.emplname) "manager"

from emp e, emp m

where e.mgrempid = m.empid

order by 1,2

;

ANSI join syntax:

select e.empid, concat(e.empfname,' ',e.emplname) "employee", e.mgrempid, concat(m.empfname,' ',m.emplname) "manager"

from emp e join emp m

on e.mgrempid = m.empid

order by 1,2

;

All above joins are called Inner joins where there will be output only if columns in WHERE clause have matching data in both tables.  
In above output, empid 1 is not displayed because it doesn't have any manager.

select \* from emp where empid = 1;

So, how to display record for that employee sans manager detail?

# Outer Joins

Outer joins are for finding records that may not have a match in the other table. You specify which side of the join is allowed to have a missing record.

**left outer join** (or just left join) -> Table on right side can have missing records  
**right outer join** (or just right join) -> Table on left side can have missing records

select

e.empid

, concat(e.empfname,' ',e.emplname) "employee"

, e.mgrempid

, concat(m.empfname,' ',m.emplname) "manager"

from emp e left outer join emp m

on e.mgrempid = m.empid

order by 1,2

;

In above example, empid 1 doesn't have manager id, so manager name will be displayed as NULL rather than not displaying row for empid 1 at all.

# Subquery

This is a query in 'where' clause of a bigger query and serves as columns clause of the 'where' clause.

**Syntax:**

select t1.col1, t1.col2... from tab\_1 t1 where t1.col1 =/in (select t2.col1, t2.col2... from tab\_2 t2 where t2.col3 = t1.col3 and....);

Example - Find out all employees who are in Mumbai location.

select \* from emp e where e.depid in (select d.depid from deptt d where d.depcity = 'Mumbai');

Inline SELECT

In this method, a column of a 'select' itself is a result of another 'select' query.

select e.\*,(select d.depcity from deptt d where d.depid = e.depid) dep\_city from emp e;

Mark as Completed

# Aggregate Functions

An aggregate function is a function where the values of multiple rows are grouped together to form a single value of more significant meaning or measurements such as a set, a bag or a list.

select max(empsal) from emp;

select avg(empsal) from emp;

select sum(empsal) from emp;

select count(empsal) from emp;

# Grouping on Aggregate Functions

Same Aggregate Functions are applied on grouped data. Grouping is done on the required columns of the table.

For example, you need to know manager wise sum of salaries of employees.

select sum(empsal) sum\_sal,mgrempid from emp

group by mgrempid

order by mgrempid

;

# HAVING clause

It works as WHERE clause for aggregate functions.

Example - You need to know city wise max salaries of employees where the max salary is more than 2150.

select max(e.empsal) max\_sal, d.depcity

from emp e, deptt d

where e.depid = d.depid

group by d.depcity

having max(e.empsal) > 2150 -- to show max sal > 2150

order by d.depcity

;

Constraint Keys on Tables - Primary and Unique Key

**Primary Key and Unique Key**

* Tables represent some real-world data and in most of the cases, one row in a table represents one real-world entity. Hence, exact data should not be repeating across the columns.  
  For example, in a table storing students information, one roll number should not occur more than once but names can repeat because more than one students can have same name.
* To enforce such a requirement, the database provides a constraint mechanism called Primary Key which is created at the table level comprising of the columns which can not repeat in the table.  
  In the current example, roll number would be the column of the Primary Key.
* Unique Key is another type of table constraint to enforce the uniqueness of the data in a table with certain differences when compared to Primary Key.

**Primary Key vs Unique Key**

* Primary Key columns cannot be the NULLABLE columns and cannot have null values while Unique Key columns can have.
* A table can have only one Primary Key but can have multiple Unique Keys.

**Auto Generation of Primary/Unique Keys**

* Primary/Unique Key values can have predefined values or can be generated using AUTO\_INCREMENT option for a column in real time as data in inserted in tables.  
  AUTO\_INCREMENT option automatically generates next number and assigns to Primary/Unique key column.
* There can be only one column with an AUTO\_INCREMENT option in a given table.  
  If AUTO\_INCREMENT is used, Primary/Unique Key must be defined as part of table creation itself using the AUTO\_INCREMENT column.  
  If AUTO\_INCREMENT is not used, Primary/Unique Key may be defined as part of table creation or may be added later.  
  Column with such an option must be added as Primary Key while creating the table.  
  You can get last inserted id value using below command and we will see examples on this later.  
  SELECT LAST\_INSERT\_ID();

Let's see all above in action.  
**With AUTO\_INCREMENT and Primary key**

use retail\_db

drop table deptt;

create table deptt (

depid int not null auto\_increment,

depname varchar(100) not null,

depcity varchar(100) not null,

depstreet varchar(100) not null,

depopendate date,

primary key ( depid )

);

**Without AUTO\_INCREMENT and with Primary key**

use retail\_db

drop table deptt;

create table deptt (

depid int not null,

depname varchar(100) not null,

depcity varchar(100) not null,

depstreet varchar(100) not null,

depopendate date,

primary key ( depid )

);

**Without AUTO\_INCREMENT and with the Primary key added later**

use retail\_db

drop table deptt;

create table deptt (

depid int not null,

depname varchar(100) not null,

depcity varchar(100) not null,

depstreet varchar(100) not null,

depopendate date

);

--system named primary key

alter table deptt add primary key (depid);

alter table deptt drop primary key;

--user named primary key

alter table deptt add primary key pk\_deptt (depid);

alter table deptt drop primary key;

* View the table definition using below commands:  
  describe deptt;  
  OR  
  show create table deptt;

describe deptt;

+-------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------------+--------------+------+-----+---------+----------------+

| depid | int(11) | NO | PRI | NULL | auto\_increment |

| depname | varchar(100) | NO | | NULL | |

| depcity | varchar(100) | NO | | NULL | |

| depstreet | varchar(100) | NO | | NULL | |

| depopendate | date | YES | | NULL | |

+-------------+--------------+------+-----+---------+----------------+

5 rows in set (0.01 sec)

Constraint Keys on Tables - Foreign Key

**Foreign Key**

* In a typical RDBMS, tables have parent/child relationships which are enforced by Referential Keys.
* Each value in referencing column of child table must exist in the referenced column of parent table.

Create table command with foreign key

use retail\_db

drop table emp;

create table emp(

empid int not null auto\_increment,

empname varchar(100) not null,

depid int not null,

mgrempid int,

primary key ( empid ),

foreign key fk\_emp\_depid (depid) references deptt(depid) on delete restrict

);

ON DELETE RESTRICT -> It prevents delete from parent table if records are present in the child table.  
Above example gives system named constraint.

Referential keys can also be added after the table creation if not added in the table creation statement.

show create table emp;

alter table emp drop foreign key emp\_ibfk\_1;

alter table emp

add constraint fk\_emp\_depid

foreign key (depid) references deptt(depid);

Above example gives a user named constraint.

**Self-referencing Foreign Key**

It is a foreign key which refers to another column in same table.  
For example, data of employees and their managers have to be kept in the same table as all are employees only at the end.  
At the same time, to represent manager for each employee, we have a column 'mgrempid' which is employee id of the manager.  
This column will refer to 'empid' of the same table which will have full details of the manager as an employees.

alter table emp

add constraint fk\_emp\_mgrempid

foreign key (mgrempid) references emp(empid);

use retail\_db;

drop table emp;

drop table deptt;

create table deptt (

depid int not null auto\_increment,

depname varchar(100) not null,

depcity varchar(100) not null,

depstreet varchar(100) not null,

depopendate date,

primary key ( depid )

);

insert into deptt (depname,depcity, depstreet,depopendate)

values('Accounts','Bengaluru','Daker Street',NOW());

insert into deptt (depname,depcity, depstreet,depopendate)

values('HR','Delhi','Baker Street',NOW());

insert into deptt (depname,depcity, depstreet,depopendate)

values('Finance','Mumbai','Laker Street',NOW());

insert into deptt (depname,depcity, depstreet,depopendate)

values('Travel','Pune','Haker Street',NOW());

create table emp(

empid int not null auto\_increment,

empfname varchar(100) not null,

emplname varchar(100) not null,

depid int not null,

doj datetime,

empsal int not null,

mgrempid int,

primary key ( empid )

,foreign key (depid) references deptt(depid) on delete restrict

);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Beck','Dot',2,'2014-09-10',2500,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Jack','Den',2,'2017-10-28',2400,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Emil','Bon',3,'2018-01-20',2100,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Siera','Leck',3,'2014-02-17',2000,4);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Koe','Hoker',3,'2016-03-16',2900,4);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Kate','Huds',4,'2017-11-22',2500,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Rene','Swaz',4,'2017-09-19',2450,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Zack','Night',4,'2013-05-15',2800,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Beck','Decker',4,'2015-06-23',2350,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Bruno','Mars',2,'1985-10-08',2350,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Kale','Hacker',2,'1984-12-17',2800,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Colt','Speed',4,'1987-11-18',2800,2);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Etal','Femme',1,'1985-01-01',2425,2);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Mont','Blanc',1,'1987-05-01',2425,2);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Iron','Man',4,'1984-12-17',2800,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Spider','Man',2,'2015-06-23',2350,3);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Captain','America',2,'2014-02-17',2000,1);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Super','Hulk',1,'2014-02-17',2000,2);

insert into emp (empfname, emplname, depid, doj, empsal,mgrempid)

values('Cat','Woman',2,'2016-05-22',2375,3);

select \* from emp where empfname like '%k%' or emplname like '%k%' -> ;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

| empid | empfname | emplname | depid | doj | empsal | mgrempid |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

| 1 | Beck | Dot | 2 | 2014-09-10 00:00:00 | 2500 | 1 |

| 2 | Jack | Den | 2 | 2017-10-28 00:00:00 | 2400 | 3 |

| 4 | Siera | Leck | 3 | 2014-02-17 00:00:00 | 2000 | 4 |

| 5 | Koe | Hoker | 3 | 2016-03-16 00:00:00 | 2900 | 4 |

| 6 | Kate | Huds | 4 | 2017-11-22 00:00:00 | 2500 | 3 |

| 8 | Zack | Night | 4 | 2013-05-15 00:00:00 | 2800 | 3 |

| 9 | Beck | Decker | 4 | 2015-06-23 00:00:00 | 2350 | 3 |

| 11 | Kale | Hacker | 2 | 1984-12-17 00:00:00 | 2800 | 1 |

| 18 | Super | Hulk | 1 | 2014-02-17 00:00:00 | 2000 | 2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

ALTER command can be used to add/modify definition of various aspects of database and its objects.<br>

**SYNTAX**

**Dropping, Adding or Repositioning a Column**<br>

--drop existing column

alter table test\_tbl drop i;

--add the same column at the end of the table

alter table test\_tbl add i int;

--drop existing column

alter table test\_tbl drop i;

--add the same column at top of the able

alter table test\_tbl add i int first;

--drop existing column

alter table test\_tbl drop i;

--add same column at after column c

alter table test\_tbl add i int after c;

**Changing a Column Definition or a Name**<br>

--change datatype

alter table test\_tbl modify col1 char(10);

--change column name

alter table test\_tbl change col1 col1new bigint;

--change datatype

alter table test\_tbl change col1new col1new int(1);

--setting the default value

alter table test\_tbl modify i bigint not null default 100;

**Changing a Column's Default Value**<br>

--setting a default value of a column

alter table test\_tbl i set default 1000;

--remove the default value of a column

alter table test\_tbl alter i drop default;

**Changing a Table Type**<br>

--change the db engine type for a given table

alter table test\_tbl type = myisam;

**Renaming a Table**<br>

--rename a table

alter table test\_tbl rename to alter\_tbl;

mysql> create table test123 (col1 int, col2 varchar(20), col3 int, col4 datetime, col5 varchar(20));

Query OK, 0 rows affected (0.09 sec)

mysql> describe test123;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| col1 | int(11) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | datetime | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+-------+-------------+------+-----+---------+-------+

5 rows in set (0.01 sec)

mysql> alter table test123 drop col4;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table test123 add col4 int;

Query OK, 0 rows affected (0.04 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| col1 | int(11) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

+-------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 drop col4;

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table test123 add col4 int first;

Query OK, 0 rows affected (0.07 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| col4 | int(11) | YES | | NULL | |

| col1 | int(11) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+-------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 drop col4;

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table test123 add col4 int after col3;

Query OK, 0 rows affected (0.06 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| col1 | int(11) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+-------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec

mysql> alter table test123 modify col1 char(10);

Query OK, 0 rows affected (0.10 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| col1 | char(10) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+-------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 change col1 colNew int;

Query OK, 0 rows affected (0.14 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| colNew | int(11) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

5 rows in set (0.01 sec)

mysql> alter table test123 change colNew colNew varchar(10);

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| colNew | varchar(10) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 alter col3 set default 1000;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| colNew | varchar(10) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | 1000 | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 alter col3 drop default;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> describe test123;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| colNew | varchar(10) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

mysql> alter table test123 rename to altertest123;

Query OK, 0 rows affected (0.04 sec)

mysql> describe test123;

ERROR 1146 (42S02): Table 'retail\_db.test123' doesn't exist

mysql> describe altertest123;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| colNew | varchar(10) | YES | | NULL | |

| col2 | varchar(20) | YES | | NULL | |

| col3 | int(11) | YES | | NULL | |

| col4 | int(11) | YES | | NULL | |

| col5 | varchar(20) | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

5 rows in set (0.01 sec)

Indexes

What are indexes?

* Indexes on a table are similar to indexes found at the end of a book to help you reach the required topics/chapters quickly without flipping through the whole book.
* Indexes are data structures storing the pointer to the actual physical location of records on the disk.
* These index data structures themselves are stored on disk to make them persistent.
* Indexes are generally created on columns which are most likely to be part of the SELECT expression, WHERE clauses and ORDER BY expression.
* If such a query is run whose all columns are present in one or more indexes, then the query may not need to touch the tables at all. Rather, the query will get all the data from indexes only.
* SELECT becomes faster because of indexes.SELECT expression.
* INSERT/UPDATE/DELETE become slower because of indexes as indexes also need to be updated accordingly in runtime.SELECT expression.

Indexes can have of 2 types of data structures:

* Btree (Ordered binary tree, defaul type) -> Good for =, !=, >=, >=, BETWEEN, LIKE, IN operators.
* Hash (Not ordered, Hash values based, more like key-value pair) -> Good for = operator.

Indexes can be -> Unique / Non-Unique

* **Unique** -> Primary key of a table or any other set of one or more columns which would identify a unique record in the table.  
  For example - Emp Id OR Customer Id and Bank Account No
* **Non-Unique** -> Any set of one or more columns to speed up the query on non-unique columns.  
  For example - Index on First name and Last name.

**Creating Indexes:**

create unique index using btree dep\_name\_idx on deptt (depname);

create unique index dep\_name\_idx on deptt (depname);

--here, name of index is given by user

create index emp\_names on emp (empfname, emplname);

--here, the name of the index will be generated by the system

alter table emp add index (empfname, emplname) using hash;

**Dropping index:**

drop index emp\_names on emp;

alter table emp drop index emp\_namess;

**Viewing indexes on a table:**

show index from emp;

Temporary Tables

* Used to store temporary data.
* Table and data are deleted when the current client session terminates.
* Not listed in the output of SHOW TABLES command.

Example:

create temporary table salessummary (

-> product\_name varchar(50) not null,

-> total\_sales decimal(12,2) not null default 0.00

);

Insert some data and select the same.

Now log out of the session or log back in. Try to select the data. You will see there is no such table or data.

Transactions in MySQL

**What is a transaction?**

* A database transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit.
* All individual operations must succeed or we must revert changes done by each operation in order to call this unit of work as a valid transaction.
* All transactions have four standard properties - **Atomicity, Consistency, Isolation, Durability (ACID)**.

**MySQL transaction features:**

* In MySQL, by default, each statement is a transaction i.e. each modification is instantly committed into the database and you can not rollback the changes made. The behavior of transaction is controlled by autocommit.
* MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error.
* A session that has autocommit enabled can perform a multiple-statement transaction by starting it with an explicit start transaction or begin statement and ending it with a commit or rollback statement.
* If autocommit mode is disabled within a session with set autocommit = 0, the session always has a transaction open. A commit or rollback statement ends the current transaction and a new one starts.
* If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.
* Some statements implicitly end a transaction, as if you had done a commit before executing the statement. For example - create, drop, alter and many others.
* **autocommit -> 1 (ON)** (enabled, default value) -> Each SQL statement (until and unless started in an explicit transaction using start transaction) is considered a complete transaction and committed by default when it finishes.
* **autocommit -> 0 (OFF)** (disabled, using command set autocommit = 0) -> Subsequent statements acts like a transaction and no activities are committed until an explicit commit statement is issued.
* If autocommit is 0 and you change it to 1, MySQL performs an automatic commit of any open transaction.
* To turn off autocommit globally for all so that clients always begin with a default of 0, we have below options: Add a autocommit=0 variable in my.cnf configuration file in MySQL and restart the MySQL server. Start the server with the --autocommit=0 option
* You can see current autocommit mode using below commands: show variables like 'autocommit'; select @@autocommit;

To illustrate, we need to transfer amount x from account A to account B.

**Sample pseudo syntax can be as follows:**

-- begin a new transaction

begin work; -- or start transaction; -- or set autocommit = 0;

A.balance = A.balance - x;

If above fails -> rollback;

B.balance = B.balance + x;

If above fails -> rollback;

Update ledger table;

If above fails -> rollback;

Send email and SMS notification to both account holders.

If above fails -> rollback;

commit; --or begin work; to start next new transaction.

-- transaction has ended.

Database Export

Export of data may be needed to have a backup of the data or to migrate data from one database to another.

**Exporting Data with the SELECT ... INTO OUTFILE Statement**

* The file is always created on the server side on MySQL host.
* Default output format is tab-delimited, linefeed-terminated file.

Example:

select \* from emp into outfile '/tmp/emp.txt';

The output format can be customized. For example, to get the output file in CSV format with CRLF-terminated lines, use below code:

select \* from emp into outfile '/tmp/emp.txt'

fields terminated by ',' enclosed by '"'

lines terminated by '\r\n';

**Exporting Tables using mysqldump program**  
mysqldump program is used to dump data into files either in raw data format or in create/insert statements format.

**Syntax:**

$ mysqldump [options] db\_name [tbl\_name ...]

$ mysqldump [options] --databases db\_name ...

$ mysqldump [options] --all-databases

**Exporting as Raw Data**

* We specify a directory where output files will be written.
* Produce tab-separated text-format data files.
* For each dumped table, mysqldump creates a tbl\_name.sql file that contains the CREATE TABLE statement that creates the table, and the server writes a tbl\_name.txt file that contains its data.
* The option value is the directory in which to write the files.

Below command will export data of emp and deptt tables from retail\_db database into /tmp directory into files emp.txt and deptt.txt, and will write 'create table' statements into files emp.sql and deptt.sql.

mysqldump -u sqoopuser -p --tab=/tmp retail\_db emp,deptt

If --no-create-info option is used, then 'create table' statements files are not created as shown below.

mysqldump -u sqoopuser -p --tab=/tmp retail\_db emp,deptt

**Exporting as Create/Insert statements**

Done using mysqldump but without **--tab** option.

Exporting certain tables:

Below will export the tables into a file which will have 'create table' statements for the tables and 'insert' statements for the data.

mysqldump -u sqoopuser -p retail\_db emp,deptt > emp\_dump.txt

Exporting whole database:

mysqldump -u sqoopuser -p retail\_db > retail\_db\_dump.txt

Exporting all databases:

mysqldump -u sqoopuser -p --all-databases > all\_database\_dump.txt

**Copying Tables or Databases to Another Host**

Export into a dump file from source database:

mysqldump -u sqoopuser -p retail\_db emp,deptt > tables\_dump.txt

Import the dump into target database:  
Create database retail\_db\_new before hand on target server.

mysql -u sqoopuser -p retail\_db\_new < tables\_dump.txt

**Export / Import directly without intermediate dump file**

If you have access to a remote MySQL database from source database server, then you can directly copy databases/tables from current server to a remote server over the network without creating any dump file.

mysqldump -u sqoopuser -p retail\_db | mysql -h remote-host.com retail\_db\_new

Database Export

Export of data may be needed to have a backup of the data or to migrate data from one database to another.

**Exporting Data with the SELECT ... INTO OUTFILE Statement**

* The file is always created on the server side on MySQL host.
* Default output format is tab-delimited, linefeed-terminated file.

Example:

select \* from emp into outfile '/tmp/emp.txt';

The output format can be customized. For example, to get the output file in CSV format with CRLF-terminated lines, use below code:

select \* from emp into outfile '/tmp/emp.txt'

fields terminated by ',' enclosed by '"'

lines terminated by '\r\n';

**Exporting Tables using mysqldump program**  
mysqldump program is used to dump data into files either in raw data format or in create/insert statements format.

**Syntax:**

$ mysqldump [options] db\_name [tbl\_name ...]

$ mysqldump [options] --databases db\_name ...

$ mysqldump [options] --all-databases

**Exporting as Raw Data**

* We specify a directory where output files will be written.
* Produce tab-separated text-format data files.
* For each dumped table, mysqldump creates a tbl\_name.sql file that contains the CREATE TABLE statement that creates the table, and the server writes a tbl\_name.txt file that contains its data.
* The option value is the directory in which to write the files.

Below command will export data of emp and deptt tables from retail\_db database into /tmp directory into files emp.txt and deptt.txt, and will write 'create table' statements into files emp.sql and deptt.sql.

mysqldump -u sqoopuser -p --tab=/tmp retail\_db emp,deptt

If --no-create-info option is used, then 'create table' statements files are not created as shown below.

mysqldump -u sqoopuser -p --tab=/tmp retail\_db emp,deptt

**Exporting as Create/Insert statements**

Done using mysqldump but without **--tab** option.

Exporting certain tables:

Below will export the tables into a file which will have 'create table' statements for the tables and 'insert' statements for the data.

mysqldump -u sqoopuser -p retail\_db emp,deptt > emp\_dump.txt

Exporting whole database:

mysqldump -u sqoopuser -p retail\_db > retail\_db\_dump.txt

Exporting all databases:

mysqldump -u sqoopuser -p --all-databases > all\_database\_dump.txt

**Copying Tables or Databases to Another Host**

Export into a dump file from source database:

mysqldump -u sqoopuser -p retail\_db emp,deptt > tables\_dump.txt

Import the dump into target database:  
Create database retail\_db\_new before hand on target server.

mysql -u sqoopuser -p retail\_db\_new < tables\_dump.txt

**Export / Import directly without intermediate dump file**

If you have access to a remote MySQL database from source database server, then you can directly copy databases/tables from current server to a remote server over the network without creating any dump file.

mysqldump -u sqoopuser -p retail\_db | mysql -h remote-host.com retail\_db\_new

Database Import

**Importing Data with LOAD DATA**

* LOAD DATA is used to import the data from the file created using the export method of SELECT ... INTO OUTFILE statement or using method of mysqldump .
* Target tables should be pre-created.

Example:

LOAD DATA LOCAL INFILE 'emp\_dump.txt' INTO TABLE table1;

LOCAL -> This keyword means dump file will be searched for on current client server. If this LOCAL keyword is omitted, then dump file will be searched for on MySQL host server.

* Default expected format of input file -> Lines terminated by linefeeds (newlines) and data values within a line separated by tabs.
* Custom options can be given according to the format of file.

For example, if the file contains values separated by colons and lines terminated by carriage returns and newline character, then use below command.

LOAD DATA LOCAL INFILE 'emp\_dump.txt' INTO TABLE emp FIELDS TERMINATED BY ':' LINES TERMINATED BY '\r\n';

* LOAD DATA expects that columns order in the file is same as in target table.
* If the order is not same, the correct list can be specified in the command.
* If your table has columns a, b, and c but successive columns in the data file correspond to columns b, c, and a, then you need to specify the column order.

Example:

LOAD DATA LOCAL INFILE 'emp\_dump.txt' INTO TABLE mytbl (b, c, a);

**Importing Data with mysqlimport**

* mysqlimport is a wrapper around LOAD DATA.
* Target tables should be pre-created.

Example:

mysqlimport -u sqoopuser -p --local db\_name dump\_emp.txt

You can also specify format.

mysqlimport -u sqoopuser -p --local --fields-terminated-by = ":" --lines-terminated-by = "\r\n" db\_name emp\_dump.txt

You can specify columns order too here.

mysqlimport -u sqoopuser -p --local --columns=b,c,a db\_name dump.txt

**Importing Data with mysql**

If the file has table create and insert statements, then use below.

mysql -u sqoopuser -p < dump.txt

# ACID Properties in DBMS

**ACID (Atomicity, Consistency, Isolation, Durability)** is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc.

In the context of databases, a sequence of database operations that satisfies the ACID properties, and thus can be perceived as a single logical operation on the data, is called a transaction.

For example, a transfer of funds from one bank account to another involves debiting from one account and crediting to another, and this whole process is a single transaction.

* **Atomicity**  
  All statements of a transaction must succeed completely, or fail completely in each and every situation, including power failures, errors and crashes. Example - Debiting and crediting in a money transfer transaction, both must happen either together or not at all.
* **Consistency**  
  The database must remain in a consistent state after any transaction. Data in the database should not have any changes other than intended after the transaction completion.
* **Isolation**  
  Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.
* **Durability**  
  Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure which actually means recording the completed transactions (or their effects) in non-volatile memory.

# Transaction Isolation Levels

Isolation Levels indicate what kind of modified/committed/uncommitted data are visible to database users to maintain the data integrity and consistency.

* MySQL uses InnoDB engine for DBMS and supports all 4 isolation levels as dictated by SQL:1992 standards: **READ UNCOMMITTED** **READ COMMITTED** **REPEATABLE READ** **SERIALIZABLE**.
* The **default isolation level** for InnoDB is **REPEATABLE READ**. which can be changed using SET TRANSACTION command.

Different RDBMSs would have one of these as default isolation level but they also provide a mechanism to switch to any required isolation level.

Isolation levels are defined in terms of below three phenomena that are either permitted or not at a given isolation level:

* **Dirty read**: The meaning of this term is as bad as it sounds. You're permitted to read uncommitted, or dirty, data. You can achieve this effect by just opening an OS file that someone else is writing and reading whatever data happens to be there. Data integrity is compromised, foreign keys are violated, and unique constraints are ignored.
* **Nonrepeatable read**: This simply means that if you read a row at time T1 and try to reread that row at time T2, the row may have changed. It may have disappeared, it may have been updated, and so on.
* **Phantom read**: This means that if you execute a query at time T1 and re-execute it at time T2, additional rows may have been added to the database, which may affect your results. This differs from a nonrepeatable read in that with a phantom read, data you already read hasn't been changed, but instead, more data satisfies your query criteria than before.

Different isolation levels are a combination of the above factors.

Isolation Level Dirty Read Nonrepeatable Read Phantom Read

READ UNCOMMITTED Permitted Permitted Permitted

READ COMMITTED -- Permitted Permitted

REPEATABLE READ -- -- Permitted

SERIALIZABLE -- --