MySQL + PYTHON CONNECTIVITY — Exam Cheat Sheet

Class XII Computer Science (083) | every command, constraint, function and connectivity template

1. Words the examiner uses

Term	Meaning
Relation	a table
Tuple / Record	one row of a table
Attribute / Field	one column of a table
Degree	number of columns (attributes) in a table
Cardinality	number of rows (tuples) in a table
Domain	the set of values a column is allowed to hold
Primary key	column (or group of columns) that uniquely identifies each row; cannot be NULL or repeated
Candidate key	any column that COULD have been the primary key
Alternate key	candidate key that was not chosen as the primary key
Foreign key	column in one table that refers to the primary key of another table
Composite key	primary key made of two or more columns together
NULL	value not known / not entered. It is NOT zero and NOT a blank space

2. The three families of SQL commands

Category	Full form	Commands	What they act on
DDL	Data Definition Language	`CREATE, ALTER, DROP, TRUNCATE`	structure of the database / table
DML	Data Manipulation Language	`INSERT, UPDATE, DELETE, SELECT`	data inside the table
TCL	Transaction Control Language	`COMMIT, ROLLBACK, SAVEPOINT`	saving or undoing changes

- Every SQL statement ends with a semicolon (;) in the MySQL prompt.
- SQL keywords are not case sensitive, but data inside quotes IS case sensitive.

3. Common data types

Type	Use	Example
`INT / INTEGER`	whole numbers	`rollno INT`
`FLOAT / DECIMAL(p,d)`	numbers with a decimal part; p = total digits, d = digits after the point	`fee FLOAT, price DECIMAL(8,2)`
`CHAR(n)`	fixed length text — always takes n characters	`grade CHAR(1)`
`VARCHAR(n)`	variable length text — takes only what is needed, up to n	`name VARCHAR(25)`
`DATE`	a date in YYYY-MM-DD form	`dob DATE`
`TIME / DATETIME`	time, or date with time	`entry DATETIME`

CHAR wastes space but is faster; VARCHAR saves space. Dates must always be written inside quotes: '2026-03-15'.

4. Database level commands

```
CREATE DATABASE school;            -- make a new database
SHOW DATABASES;                    -- list all databases
USE school;                        -- open a database for work
DROP DATABASE school;              -- delete the database completely
```

5. Table level commands (DDL)

```
CREATE TABLE student (
    rno   INT PRIMARY KEY,
    name  VARCHAR(25) NOT NULL,
    dob   DATE,
    fee   FLOAT DEFAULT 0 );

SHOW TABLES;                       -- list tables of the open database
DESCRIBE student;   or   DESC student;    -- show the structure
```

```
-- ALTER TABLE : change the STRUCTURE
ALTER TABLE student ADD marks INT;                -- add a column
ALTER TABLE student MODIFY name VARCHAR(40);      -- change type/size
ALTER TABLE student CHANGE name sname VARCHAR(40); -- rename a column
ALTER TABLE student DROP marks;                   -- remove a column
ALTER TABLE student ADD PRIMARY KEY (rno);        -- add a constraint
ALTER TABLE student DROP PRIMARY KEY;             -- remove it

DROP TABLE student;         -- deletes structure AND data
TRUNCATE TABLE student;     -- deletes all rows, structure stays
```

Command	Removes data	Removes structure	Can be rolled back
`DELETE FROM t;`	yes	no	yes (DML)
`TRUNCATE TABLE t;`	yes	no	no (DDL)
`DROP TABLE t;`	yes	yes	no (DDL)

6. Constraints

Constraint	What it enforces
`NOT NULL`	the column can never be left empty
`UNIQUE`	no two rows may hold the same value; NULL is allowed
`PRIMARY KEY`	UNIQUE + NOT NULL together; only one per table
`FOREIGN KEY`	the value must already exist in the primary key of another table
`DEFAULT value`	if no value is given, this value is used
`CHECK (condition)`	the value must satisfy the condition
`AUTO_INCREMENT`	MySQL fills the number itself, increasing by 1 each time

```
CREATE TABLE emp (
    empid  INT PRIMARY KEY AUTO_INCREMENT,
    name   VARCHAR(30) NOT NULL,
    email  VARCHAR(40) UNIQUE,
    salary FLOAT CHECK (salary > 0),
    city   VARCHAR(20) DEFAULT "Delhi",
    deptid INT,
    FOREIGN KEY (deptid) REFERENCES dept(deptid) );
```

- A table can have many UNIQUE and many FOREIGN KEY constraints, but only ONE primary key.
- The table referred to by a foreign key is the parent table; it must be created first.

7. Putting data in and changing it (DML)

```
-- INSERT : add a row
INSERT INTO student VALUES (1, "Aarav", "2008-05-14", 5400);
INSERT INTO student (rno, name) VALUES (2, "Siya");   -- selected columns

-- UPDATE : change existing rows  (WHERE decides which rows)
UPDATE student SET fee = 6000 WHERE rno = 1;
UPDATE student SET fee = fee + 500 WHERE fee < 5000;

-- DELETE : remove rows
DELETE FROM student WHERE rno = 2;
DELETE FROM student;            -- removes every row
```

Leaving out WHERE in UPDATE or DELETE changes or deletes EVERY row. This is the most common mistake in the paper.

8. SELECT — the query command

```
SELECT * FROM student;                      -- all columns, all rows
SELECT name, fee FROM student;              -- chosen columns
SELECT DISTINCT city FROM student;          -- remove duplicate values
SELECT name, fee*12 AS "Annual Fee" FROM student;   -- column alias
SELECT * FROM student WHERE fee > 5000;     -- condition
SELECT * FROM student ORDER BY fee DESC;    -- sort, ASC is default
SELECT * FROM student LIMIT 3;              -- first 3 rows only
```

Order of the clauses — never change it

```
SELECT   column list
FROM     table
```

```
WHERE    row condition
GROUP BY column
HAVING   group condition
ORDER BY column
LIMIT    n;
```

9. Operators used in WHERE

Operator	Meaning	Example
= , > , < , >= , <= , <>	comparison (<> means not equal)	WHERE fee <> 5000
AND , OR , NOT	combine conditions	WHERE fee > 5000 AND city = "Delhi"
BETWEEN ... AND ...	range, both ends included	WHERE fee BETWEEN 4000 AND 6000
IN (list)	matches any value of the list	WHERE city IN ("Delhi","Agra")
LIKE	pattern matching	WHERE name LIKE "A%"
IS NULL / IS NOT NULL	test for an empty value	WHERE dob IS NULL

Wildcard	Meaning	Example	Matches
%	any number of characters (even none)	"A%"	Aarav, Ananya, A
_	exactly one character	"_a%"	Rahul, Kabir — second letter is a
combined	both together	"%a_"	any name whose second-last letter is a

Never write = NULL or <> NULL. NULL can only be tested with IS NULL and IS NOT NULL.

10. Aggregate functions, GROUP BY and HAVING

Function	Returns	Counts NULL?
COUNT(*)	number of rows	yes
COUNT(column)	number of NON-NULL values in that column	no
SUM(column)	total	no
AVG(column)	average	no
MAX(column) / MIN(column)	largest / smallest value	no

```
SELECT COUNT(*) FROM student;
SELECT AVG(fee), MAX(fee) FROM student;

-- GROUP BY : one answer per group
SELECT city, COUNT(*) FROM student GROUP BY city;
SELECT city, AVG(fee) FROM student GROUP BY city;

-- HAVING : condition ON THE GROUPS
SELECT city, COUNT(*) FROM student
GROUP BY city HAVING COUNT(*) > 2;

-- WHERE first, then grouping, then HAVING
SELECT city, AVG(fee) FROM student
WHERE fee > 3000 GROUP BY city HAVING AVG(fee) > 5000;
```

	WHERE	HAVING
Works on	individual rows	groups made by GROUP BY
Used	before grouping	after grouping
Aggregate function allowed	no	yes

Any column written in the SELECT list along with an aggregate function must also appear in GROUP BY.

11. Working with two tables — joins

```
-- Cartesian product : every row of A with every row of B
SELECT * FROM student, dept;
-- degree = sum of degrees, cardinality = product of cardinalities
```

```
-- Equi join : rows matched on a common column (use this one)
SELECT student.name, dept.dname
FROM student, dept
WHERE student.deptid = dept.deptid;

-- The same query written with JOIN ... ON
SELECT s.name, d.dname
FROM student s JOIN dept d ON s.deptid = d.deptid;

-- Natural join : the common column appears only once
SELECT * FROM student NATURAL JOIN dept;
```

- When the same column name exists in both tables, write it as tablename.column, otherwise MySQL reports an ambiguous column.
- A join of two tables always needs a joining condition, otherwise you get the cartesian product.

12. Python — MySQL connectivity

The five steps, every time

```
import mysql.connector                          # 1. import the module

con = mysql.connector.connect(host="localhost",# 2. open a connection
                              user="root",
                              passwd="tiger",
                              database="school")

cur = con.cursor()                              # 3. create a cursor
cur.execute("SELECT * FROM student")            # 4. run the query
data = cur.fetchall()
for rec in data:
    print(rec)

con.close()                                     # 5. close the connection
```

- The module is installed once from the command prompt: pip install mysql-connector-python
- It is usually imported as import mysql.connector as mycon so that the connect line becomes shorter.
- con.is_connected() returns True if the connection was made — useful to check your host, user and password.

Statement	What it does	Returns
`con = mysql.connector.connect(...)`	opens the connection to MySQL	connection object
`cur = con.cursor()`	creates the cursor that carries queries	cursor object
`cur.execute(query)`	sends one SQL query to MySQL	nothing
`cur.fetchone()`	the NEXT single record	one tuple, or None
`cur.fetchall()`	all remaining records	list of tuples
`cur.fetchmany(n)`	the next n records	list of tuples
`cur.rowcount`	how many rows the last query touched	integer (-1 if none)
`con.commit()`	SAVES the changes permanently	nothing
`con.close()`	closes the connection	nothing

commit() is compulsory after INSERT, UPDATE and DELETE. Without it the change is lost when the program ends. It is NOT needed after SELECT.

- The cursor moves forward only. After fetchall() a second fetchall() returns an empty list — run execute() again to read once more.
- fetchone() returns one tuple; fetchall() returns a list of tuples even when only one record matched.

Writing the query with values from the user

```
# Method 1 : format()  — numbers plain, text inside quotes
query = "INSERT INTO student VALUES ({},'{}','{}',{})"
query = query.format(rno, name, dob, fee)
cur.execute(query)

# Method 2 : placeholders (safer, no quoting mistakes)
```

```
query = "INSERT INTO student VALUES (%s, %s, %s, %s)"
cur.execute(query, (rno, name, dob, fee))

# Method 3 : f-string
cur.execute(f"SELECT * FROM student WHERE fee > {amount}")
```

Numbers take {} with no quotes. Strings and dates take '{}' with single quotes around them. Forgetting these quotes is the usual reason a program fails.

13. The four connectivity templates

(a) INSERT — add a record entered by the user

```
import mysql.connector as mycon
con = mycon.connect(host="localhost", user="root",
                    passwd="tiger", database="school")
cur = con.cursor()
rno  = int(input("Enter Roll Number: "))
name = input("Enter Name: ")
query = "INSERT INTO student VALUES ({},'{}')".format(rno, name)
cur.execute(query)
con.commit()                        # compulsory
print("Data added successfully")
con.close()
```

(b) SELECT — display records that satisfy a condition

```
cur.execute("SELECT * FROM student WHERE fee > 5000")
data = cur.fetchall()
for rec in data:
    print(rec)
print("Total records :", cur.rowcount)      # no commit needed
```

(c) UPDATE — change a value

```
query = "UPDATE student SET fee = 6000 WHERE rno = 1"
cur.execute(query)
con.commit()
print(cur.rowcount, "record(s) updated")
```

(d) DELETE — remove a record

```
query = "DELETE FROM student WHERE rno = {}".format(rno)
cur.execute(query)
con.commit()
print("Record deleted")
```

14. Solved programs

Q1. Insert a record into table Student of database SCHOOL

Fields: rno (integer), name (string), DOB (date), fee (float). Host localhost, user root, password tiger. All four values are to be accepted from the user.

```
import mysql.connector as mysql

con1 = mysql.connect(host="localhost", user="root",
                     password="tiger", database="school")
mycursor = con1.cursor()

rno  = int(input("Enter Roll Number :: "))
name = input("Enter the name :: ")
DOB  = input("Enter date of birth (YYYY-MM-DD) :: ")
fee  = float(input("Enter Fee :: "))

query = "INSERT INTO student VALUES ({},'{}','{}',{})"
query = query.format(rno, name, DOB, fee)
mycursor.execute(query)
con1.commit()
print("Data added successfully")
con1.close()
```

- rno and fee are numbers, so {} is written without quotes. name and DOB are text, so '{}' carries single quotes.

Q2. Display the records of students whose fee is more than 5000

```
import mysql.connector as mysql

con1 = mysql.connect(host="localhost", user="root",
                     password="tiger", database="school")
mycursor = con1.cursor()

query = "SELECT * FROM student WHERE fee > {}".format(5000)
```

```
mycursor.execute(query)
data = mycursor.fetchall()
for rec in data:
    print(rec)
con1.close()
```

- No commit() here — SELECT only reads, it changes nothing.

Q3. AddAndDisplay() for table STATIONERY in database ITEMDB

Structure: itemNo int(11), itemName varchar(15), price float, qty int(11). Input an item, store it, then display every record whose price is more than 120. Host localhost, user root, password Pencil.

```
import mysql.connector as mycon

def AddAndDisplay():
    mydb = mycon.connect(host="localhost", user="root",
                         passwd="Pencil", database="ITEMDB")
    mycur = mydb.cursor()

    no  = int(input("Enter Item Number: "))
    nm  = input("Enter Item Name: ")
    pr  = float(input("Enter price: "))
    qty = int(input("Enter qty: "))

    query = "INSERT INTO stationery VALUES ({},'{}',{},{})"
    query = query.format(no, nm, pr, qty)
    mycur.execute(query)
    mydb.commit()

    mycur.execute("SELECT * FROM stationery WHERE price > 120")
    for rec in mycur.fetchall():
        print(rec)
    mydb.close()

AddAndDisplay()
```

- One function does two jobs here: the insert needs commit(), the display does not.

Q4. Change the Quantity to 91 where Item_code is 208

Table product_inventory in database WarehouseDB — Item_code (integer), Product_name (string), Quantity (integer), Cost (integer). Host localhost, user admin_user, password warehouse2024.

```
import mysql.connector

connection = mysql.connector.connect(host="localhost",
                                     user="admin_user",
                                     password="warehouse2024",
                                     database="WarehouseDB")
cursor = connection.cursor()

update_query = "UPDATE product_inventory SET Quantity = 91 \
WHERE Item_code = 208"
cursor.execute(update_query)
connection.commit()
print("Data updated successfully.")

cursor.close()
connection.close()
```

- The password must be typed exactly as given, with no space inside it.

15. Marks are lost here — check every time

In SQL

- Leaving out WHERE in UPDATE or DELETE — every row changes.
- Writing = NULL instead of IS NULL.
- Putting an aggregate function in WHERE. Conditions on groups go in HAVING.
- Forgetting quotes around text and dates: name = 'Aarav', dob = '2008-05-14'.
- Mixing up DELETE, TRUNCATE and DROP — only DROP removes the structure.
- Joining two tables without a joining condition, which gives the cartesian product.
- Writing the clauses out of order. SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY.

In Python connectivity

- Forgetting con.commit() after INSERT, UPDATE or DELETE — the change never reaches the database.
- Missing quotes around string values inside the query built with format().

- Using fetchall() and then trying to read the same data a second time without running execute() again.
- Writing mysql.connect(...) when the module was imported as import mysql.connector — either import it as a short name, or call mysql.connector.connect(...).
- Forgetting con.close() at the end, or forgetting to call the function.
- Not converting input(): input() gives a string, so int() or float() is needed for numeric fields.