Basics of Database Development

Exercises Answers

No.	Question	Correct Answer
1	What is a database?	**b) A systematic collection of data supporting electronic storage and manipulation.**
2	Which is NOT an advantage of a Relational DB over a Traditional File System?	**c) Always faster performance, regardless of table count.**
3	In the Relational Data Model, what is a 'Tuple'?	**c) A single row containing a single record.**
4	Which language is used to define the database structure (CREATE, ALTER, DROP)?	**c) Data Definition Language (DDL).**
5	Which key is the minimal set of fields that uniquely identify each record?	**b) Candidate Key.**
6	What does 'Physical Data Independence' mean?	**b) Changes in physical storage do not affect the logical structure.**
7	In an ER Diagram, what shape represents an Attribute?	**c) Ellipse (Oval).**
8	Student assigned to many projects; project assigned to many students — what relationship?	**d) Many to Many.**
9	What is the purpose of the SQL UNION operator?	**c) Combines result sets of two or more SELECT statements, removing duplicates by default.**
10	Why install a standalone MySQL server even with XAMPP?	**b) To practice advanced DB administration outside the web server environment.**
11	Which users interact directly with SQL but don't write full application code?	**a) Sophisticated Users.**
12	Which SQL command removes all data without deleting the table structure?	**c) TRUNCATE.**
13	Which TCL command undoes changes to maintain data integrity after an error?	**b) ROLLBACK.**
14	Who introduced the relational model in 1970?	**b) Edgar Frank 'Ted' Codd.**
15	In the Relational Model, what is a 'Domain'?	**b) A set of values permitted for an attribute in a table.**

No.	Question	Correct Answer
16	An entity that cannot be identified by its own attributes and relies on another is called a:	**c) Weak Entity.**
17	A customer places many orders, but an order belongs to one customer — which cardinality?	**b) One to Many.**
18	An entity relying on another for identification is called a: (repeat Q16)	**c) Weak Entity.**
19	How is a Derived Attribute (e.g. Age from DOB) shown in an ER diagram?	**a) Dashed oval.**
20	Customer places many orders but order belongs to one customer — cardinality? (repeat Q17)	**b) One to Many.**
21	Which hardware component is most critical for caching data (Buffer Pool)?	**d) Memory (RAM).**
22	What refers to 'Data about data' in a DBMS?	**a) Metadata.**
23	XAMPP includes which DB, so a separate MySQL install may not be needed?	**b) MariaDB.**
24	Which command ensures newly granted privileges take effect immediately?	**c) FLUSH PRIVILEGES.**
25	The INSERT INTO ... SELECT statement is used to:	**c) Insert records from another existing table.**
26	Which operator filters for a range of values in a WHERE clause?	**a) BETWEEN.**
27	What keyword temporarily renames a column in query output?	**c) AS (ALIAS).**
28	Which SQL clause is used to sort the result set?	**c) ORDER BY.**
29	Which JOIN returns only records with matching values in both tables?	**c) INNER JOIN.**
30	Which operator combines result sets of two SELECT statements and KEEPS duplicates?	**b) UNION ALL.**
31	Which operator returns only rows common to both SELECT statements?	**a) INTERSECT.**
32	In a subquery, which query executes first?	**b) The inner query (subquery).**
33	Which operator returns TRUE if a subquery returns at least one row?	**c) EXISTS.**
34	In the Relational Data Model, a Tuple is equivalent to a:	**a) A Row.**
35	Which TCL command sets a point in a transaction to roll back to later?	**b) SAVEPOINT.**
36	What type of NoSQL database is MongoDB?	**d) Document-oriented.**

No.	Question	Correct Answer
37	What does the acronym ACID stand for?	**b) Atomicity, Consistency, Isolation, Durability.**
38	Which command deletes the entire table or database structure?	**c) DROP.**
39	Which is NOT an advantage of the Relational Model?	**c) Free and low cost of physical storage.**
40	In ER Modeling, a double rectangle represents a:	**b) Weak Entity.**

Q18 & Q27 — First Name, Last Name, Email — Ordered by Last Name

Retrieve student contact details sorted alphabetically by last name using ORDER BY.

```
SELECT first_name, last_name, email
FROM students
ORDER BY last_name ASC;
```

Q19 & Q28 — Students with a Gmail Account

Use the LIKE wildcard with '%@gmail.com' to match any email ending in the Gmail domain.

```
SELECT *
FROM students
WHERE email LIKE '%@gmail.com';
```

Q20 & Q29 — Students Enrolled in 2023 (BETWEEN + LIMIT)

Filter enrollment dates using BETWEEN, then cap results with LIMIT 5.

```
SELECT *
FROM students
WHERE enrollment_date BETWEEN '2023-01-01' AND '2023-12-31'
LIMIT 5;
```

Q21 & Q30 — Students Linked to Their Courses (JOIN)

Join Students -> Enrollments -> Courses to display each student next to their course name.

```
SELECT s.first_name, s.last_name, c.course_name
FROM students s
JOIN enrollments e  ON s.student_id = e.student_id
JOIN courses c      ON e.course_id  = c.course_id;
```

Q22 & Q31 — Courses Without an Instructor (IS NULL)

Find courses where the instructor_id column is NULL, meaning no instructor is assigned yet.

```
SELECT *
FROM courses
WHERE instructor_id IS NULL;
```

Q23 & Q32 — Students Enrolled in 'Advanced PHP OOP' (Subquery + IN)

The inner subquery fetches course IDs for the target course; the outer query returns matching students.

```
SELECT first_name, last_name
FROM students
WHERE student_id IN (
    SELECT e.student_id
    FROM enrollments e
    JOIN courses c ON e.course_id = c.course_id
    WHERE c.course_name = 'Advanced PHP OOP'
);
```

Q24 — Insert Ten New Students

Use a single INSERT INTO with multiple value rows for efficiency.

```
INSERT INTO students (first_name, last_name, email, enrollment_date)
VALUES
  ('Amina',   'Uwase',      'amina.uwase@email.com',      '2024-01-10'),
  ('Jean',    'Habimana',   'jean.habimana@email.com',    '2024-01-10'),
  ('Grace',   'Mukamana',   'grace.mukamana@email.com',   '2024-01-11'),
  ('Patrick', 'Niyonzima',  'patrick.niyo@email.com',     '2024-01-12'),
  ('Diane',   'Ingabire',   'diane.inga@email.com',       '2024-01-12'),
  ('Claude',  'Bizimana',   'claude.bizi@email.com',      '2024-01-13'),
  ('Solange', 'Nyiransabimana', 'solange.n@email.com',    '2024-01-13'),
  ('Eric',    'Ndayisaba',  'eric.ndayi@email.com',       '2024-01-14'),
  ('Marie',   'Uwitonze',   'marie.uwit@email.com',       '2024-01-15'),
  ('Kevin',   'Nsengiyumva','kevin.nsen@email.com',       '2024-01-15');
```

Q25 — Update Uwimana Sifa's Phone Number

Use UPDATE with a precise WHERE clause to target only this student's record.

```
UPDATE students
SET phone number = '+250788123456'
WHERE first name = 'Sifa'
  AND last_name  = 'Uwimana';
```

Tip: Using the primary key (student_id) is safer if known, e.g. WHERE student_id = 42

Q26 — Delete the 'Introduction to Php' Course

Use DELETE with a WHERE clause to remove only the specified course row.

```
DELETE FROM courses
WHERE course_name = 'Introduction to Php';
```

Q33 — Crowded Classes (COUNT + GROUP BY + HAVING)

Group enrollments by course, count students per course, then filter for courses exceeding 20 students.

```
SELECT c.course_name, COUNT(e.student_id) AS total_students
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.student_id) > 20
ORDER BY total_students DESC;
```

Section C — Practical Query Writing (Q34–Q45)

Q34 — School Management System

Tables: students(student_id, full_name, class_id) | classes(class_id, class_name) | teachers(teacher_id, teacher_name) | subjects(subject_id, subject_name, teacher_id) | marks(marks_id, student_id, subject_id, score)

a) Add a New Student — Ineza Emmanuella (student_id 100, class_id 5)

```
INSERT INTO students (student_id, full_name, class_id)
VALUES (100, 'Ineza Emmanuella', 5);
```

b) Move All Students from class_id 5 to class_id 6

```
UPDATE students
SET class_id = 6
WHERE class_id = 5;
```

c) Students with Their Class Name (INNER JOIN — assigned only)

INNER JOIN excludes students who do not have a class assigned.

```
SELECT s.full_name, c.class_name
FROM students s
INNER JOIN classes c ON s.class_id = c.class_id;
```

d) ALL Subjects with Teacher Names (LEFT JOIN — including unassigned)

LEFT JOIN ensures subjects without a teacher still appear, with NULL in the teacher column.

```
SELECT s.subject name, t.teacher name
FROM subjects s
LEFT JOIN teachers t ON s.teacher_id = t.teacher_id;
```

e) Students Who Have Received Marks (Subquery + IN)

```
SELECT full_name
FROM students
WHERE student_id IN (
    SELECT DISTINCT student_id
    FROM marks
);
```

f) Create User 'teacher_admin' and Grant Privileges on the marks Table

Step 1 — create the user; Step 2 — grant specific privileges.

```
-- Step 1: Create the user
CREATE USER 'teacher_admin'@'localhost' IDENTIFIED BY 'pass123';

-- Step 2: Grant SELECT, INSERT, UPDATE on marks only
GRANT SELECT, INSERT, UPDATE
    ON school_db.marks
    TO 'teacher_admin'@'localhost';
```

```
-- Step 3: Apply privileges immediately
FLUSH PRIVILEGES;
```

Q35–Q41 — School System 2

Tables: students(student_id, full_name, class_id) | classes(class_id, class_name) | teachers(teacher_id, teacher_name)

Q36 — DDL: Create Database school_db

```
CREATE DATABASE school_db;
```

Q37 — DCL: Grant ALL Privileges to 'admin'@'localhost'

Best practice: create the user first, then grant.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'secret123';
GRANT ALL PRIVILEGES ON school_db.* TO 'admin'@'localhost';
FLUSH PRIVILEGES;
```

Q38 — DML: INSERT Student 'Paul' (student_id 10, class_id 2)

```
INSERT INTO students (student_id, full_name, class_id)
VALUES (10, 'Paul', 2);
```

Q39 — DML: UPDATE class_id to 3 for student_id 10

```
UPDATE students
SET class_id = 3
WHERE student_id = 10;
```

Q40 — Retrieve All Student Names Sorted Z–A (Descending)

```
SELECT full_name
FROM students
ORDER BY full_name DESC;
```

Q41 — INNER JOIN: Student Names with Their Class Names

```
SELECT s.full_name, c.class_name
FROM students s
INNER JOIN classes c ON s.class_id = c.class_id;
```

Section C — Theory Questions (Q42–Q45)

Q42 — Four Common Types of NoSQL Databases

- Document-Oriented — stores data as JSON/BSON documents (e.g. MongoDB).
- Key-Value Store — data stored as simple key-value pairs (e.g. Redis).
- Wide-Column Store — organizes data in columns rather than rows (e.g. Apache Cassandra).
- Graph Database — stores data as nodes and edges for relationship-heavy data (e.g. Neo4j).

Q43 — Logical vs Physical Data Independence

Logical Data Independence: The ability to change the logical (conceptual) schema — such as adding or removing tables and columns — without requiring changes to external views or application programs. For example, splitting one table into two should not break existing queries that reference those fields.

Physical Data Independence: The ability to change the internal (physical) storage structure — such as moving data from an HDD to an SSD, changing file organisation, or adding indexes — without affecting the logical schema or the applications above it.

In short: Logical independence protects applications from schema changes; Physical independence protects the schema from storage changes.

Q44 — DROP vs TRUNCATE

DROP: Completely removes the table (or database) and its entire structure — including all data, indexes, constraints, and the table definition itself — from the database. The operation cannot be rolled back in most DBMS.

TRUNCATE: Removes ALL rows from a table but keeps the table structure (columns, constraints, indexes) intact. It is faster than DELETE for large tables because it does not log individual row deletions. Like DROP, it generally cannot be rolled back.

Key difference: DROP deletes the table itself; TRUNCATE only empties it.

Q45 — UNION vs UNION ALL

UNION: Combines the result sets of two or more SELECT statements into a single result set and automatically removes duplicate rows. It performs a DISTINCT operation internally, which adds processing overhead.

UNION ALL: Also combines result sets but keeps ALL rows, including duplicates. Because it skips the deduplication step, it is faster than UNION and should be preferred when duplicates are acceptable or when you know the data sets are already distinct.

```
-- UNION removes duplicates
SELECT city FROM customers
UNION
SELECT city FROM suppliers;

-- UNION ALL keeps duplicates (faster)
SELECT city FROM customers
UNION ALL
SELECT city FROM suppliers;
```