🎓 MyWebUniversity.com

SQL & Databases

Complete 2-Hour Master Course

- ✅ StudentDB & AmazonShoppingDB — full build from scratch
- ✅ DDL · DML · DCL — every statement with real examples
- ✅ Normalization 1NF → 2NF → 3NF → BCNF
- ✅ MySQL & Oracle — parallel syntax throughout
- ✅ Backup: RMAN · mysqldump · Data Pump
- ✅ APIs: Python · R · Java · C++ with real connection code

📋 Course Outline — 2-Hour Master Course

Sections 01–11 with hands-on MySQL & Oracle examples throughout

01 — **SQL & DBMS Fundamentals**
Databases, types, architecture, SGA, daemons

02 — **Keys & Normalization**
PK/FK/SK/AK/CK + 1NF,2NF,3NF,BCNF

03 — **StudentDB — Full Build**
CREATE DB, users, tablespace, tables, indexes

04 — **AmazonShoppingDB — Full Build**
E-commerce schema, normalized, indexed

05 — **DDL Deep Dive**
ALTER, DROP, TRUNCATE, RENAME, views

06 — **DML Deep Dive**
INSERT,UPDATE,DELETE,SELECT,JOINs,subqueries

07 — **DCL & Security**
GRANT, REVOKE, roles, profiles, row-level sec

08 — **Backup & Recovery**
RMAN, mysqldump, Data Pump, export/import

09 — **Python/R/Java/C++ APIs**
Real code for connecting and CRUD operations

10 — **Thank You & Resources**
Channels, certifications, career resources

SECTION 01

SQL & DBMS Fundamentals

What is a Database · DBMS · Architecture · Daemons · SGA · Buffer Pool

What is a Database & DBMS?

The foundation of every modern application

Database

An organized, structured collection of data stored electronically. Every modern app — banking, shopping, social media — runs on a database.

Real examples: Your bank account, Amazon orders, Gmail inbox, Netflix history — ALL databases.

DBMS

The software that manages the database — creating, reading, updating, deleting data while enforcing security, integrity, backups, and concurrency.

MySQL · Oracle · PostgreSQL · MS SQL Server · MS Access

ACID — The 4 Rules of Reliable Databases

A — ATOMICITY
All operations succeed OR all roll back. No partial commits.

C — CONSISTENCY
DB always moves between valid states. All constraints enforced before & after.

I — ISOLATION
Concurrent transactions cannot interfere with each other. Locking/MVCC.

D — DURABILITY
Once committed, data survives crashes, power loss, server failure. Written to disk.

Types of Databases

Six major categories — choose the right tool for the right job

Relational (RDBMS)

Tables+rows+SQL+ACID. The gold standard for business data.
MySQL, Oracle, PostgreSQL, MS SQL, MS Access

Object-Oriented

Stores objects like OOP classes. db4o, ObjectDB.
Best for CAD, simulations, complex object graphs

Hierarchical/LDAP

Parent→Child tree. IBM IMS, Windows Active Directory.
LDAP on port 389: ldapsearch, ldapadd commands

NoSQL / Document

Schema-less JSON/BSON. Horizontal scale.
MongoDB, Cassandra, Redis, CouchDB, DynamoDB

NewSQL / Cloud

Relational + web scale. Google Spanner, CockroachDB,
Amazon Aurora — global ACID at massive scale

Time-Series / Graph

InfluxDB for IoT metrics, Neo4j for social graphs.
TimescaleDB — specialized high-performance engines

Oracle Architecture — SGA & Background Processes

System Global Area: the shared memory heart of Oracle

Shared Pool

Library cache (parsed SQL + exec plans) + Data Dictionary Cache. Most-tuned component in OLTP.

DB Buffer Cache

Data block cache from datafiles. LRU eviction. Dirty blocks written by DBWn. innodb_buffer_pool_size in MySQL.

Redo Log Buffer

Circular buffer logging ALL changes. Written by LGWR on COMMIT, 1/3 full, or 3-second timeout.

Large Pool

RMAN backup/restore, parallel query UGA, shared server sessions. Separate from shared pool.

Java Pool

Memory for Java stored procedures inside Oracle JVM. Usually 32MB default.

Streams Pool

Oracle Streams, GoldenGate replication, Advanced Queuing. Usually 0 unless Streams configured.

Database Daemons & Background Processes

Oracle + MySQL — always running, silently keeping your DB healthy

DBWn (DB Writer)

Writes dirty buffer cache blocks to datafiles. Triggered by checkpoint, 60s timeout, or buffer threshold.

LGWR (Log Writer)

Writes redo log buffer → online redo logs on COMMIT, 1/3 full buffer, or 3 seconds.

CKPT (Checkpoint)

Updates SCN in control file & datafile headers. Signals DBWn to flush dirty buffers to disk.

SMON (Sys Monitor)

Instance recovery at startup. Cleans temp segments. Coalesces free extents in tablespaces.

PMON (Proc Monitor)

Cleans up dead sessions. Releases locks and resources. Registers DB with Oracle Listener.

ARCn (Archiver)

Copies full online redo log groups to archive destination. Required for RMAN backup.

mysqld (MySQL)

The entire MySQL server in one daemon process. Multi-threaded. Listens on TCP 3306 by default.

InnoDB I/O Threads

Background read/write threads for asynchronous I/O. Tune with innodb_read_io_threads.

Purge Thread

InnoDB MVCC cleanup: removes old undo log records for committed transactions.

SECTION 02

Keys & Normalization

PK · FK · SK · AK · CK · 1NF · 2NF · 3NF · BCNF with Real Examples

All Database Keys — Complete Reference

Six key types with SQL syntax for both MySQL and Oracle

PK

Primary Key

Uniquely identifies every row. NOT NULL. One per table.
MySQL: id INT AUTO_INCREMENT PRIMARY KEY
Oracle: id NUMBER GENERATED ALWAYS AS IDENTITY

FK

Foreign Key

Links child to parent PK. Enforces referential integrity.
FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
ON DELETE CASCADE | SET NULL | RESTRICT

SK

Surrogate Key

System-generated, no business meaning. Best practice for PKs.
MySQL: AUTO_INCREMENT
PostgreSQL: SERIAL Oracle: SEQUENCE + NEXTVAL

AK

Alternate Key

Unique candidate key NOT chosen as PK. Enforced with:
ALTER TABLE students ADD UNIQUE (email)
CREATE UNIQUE INDEX idx_email ON students(email)

CK

Candidate Key

Any column/combo that COULD be the PK.
Must be Unique + NOT NULL.
Example: both student_id AND email qualify

CO

Composite Key

PK made of 2+ columns. Used in junction tables.
PRIMARY KEY (order_id, product_id)
Natural when no single column is unique alone

Normalization — 1NF, 2NF, 3NF, BCNF

Eliminate redundancy · Prevent anomalies · Ensure integrity

1NF — First Normal Form

Rules:
- Atomic (indivisible) values only
- No repeating groups or arrays
- Each row uniquely identifiable

❌ BEFORE:
phone: '555-1111, 555-2222'

✅ AFTER:
Create phones table with FK — one phone per row

2NF — Second Normal Form

Rules:
- Must satisfy 1NF first
- Every non-key column depends on WHOLE PK (no partial dependency)

❌ BEFORE:
order_items(order_id, prod_id, prod_name)
prod_name depends only on prod_id

✅ AFTER:
Move prod_name to products table

3NF — Third Normal Form

Rules:
- Must satisfy 2NF first
- No transitive dependencies (non-key col depends ONLY on PK)

❌ BEFORE:
emp(emp_id, dept_id, dept_name)
dept_name → dept_id not emp_id

✅ AFTER:
Move dept_name to departments table

BCNF — Boyce-Codd NF

Stricter than 3NF. For every X→Y, X must be a superkey.

❌ BEFORE:
(student,course,teacher)
teacher→course but teacher is not a superkey

✅ AFTER:
teaching(teacher,course)
enrollment(student,teacher)

Eliminate ALL functional dependency anomalies

SECTION 03

StudentDB — Full Build

MySQL & Oracle · DB · Users · Tablespace · Normalized Tables · Indexes

StudentDB — Normalized Schema (3NF/BCNF)

8 tables · proper PKs/FKs · fully normalized · ready for production

Table	PK	Key Columns	Relationships
departments	dept_id	dept_name, location, budget	→ programs, instructors
programs	prog_id	prog_name, dept_id(FK), credits, duration	→ courses, students
instructors	instructor_id	first/last_name, email(UQ), dept_id(FK), salary	→ courses
students	student_id	first/last_name, email(UQ), prog_id(FK), gpa, status	→ enrollments, grades, payments
courses	course_id	course_name, prog_id(FK), instructor_id(FK), credits	→ enrollments, grades
enrollments	(student_id,course_id,semester,year)	enrollment_date, status	M:N junction: students↔courses
grades	grade_id	student_id(FK), course_id(FK), grade_letter, grade_pts	→ GPA calculation
payments	payment_id	student_id(FK), amount, method, status, paid_at	→ financial records

StudentDB — MySQL: Create Database & Users

Step-by-step: database creation, users, privileges

MySQL — Create Database & Users

```
-- 1. Create the StudentDB database
CREATE DATABASE IF NOT EXISTS StudentDB
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

-- 2. Create application user (full access)
CREATE USER 'student_app'@'localhost'
  IDENTIFIED BY 'StrongPass!2024';

GRANT ALL PRIVILEGES ON StudentDB.*
  TO 'student_app'@'localhost';
```

Oracle — Tablespace & Schema User

```
-- 1. Create dedicated tablespace
CREATE TABLESPACE students_ts
  DATAFILE '/u01/oradata/ORCL/students01.dbf'
  SIZE 100M AUTOEXTEND ON
  NEXT 10M MAXSIZE 500M
  EXTENT MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO;

-- 2. Create temp tablespace
CREATE TEMPORARY TABLESPACE students_temp
  TEMPFILE '/u01/oradata/ORCL/stud_temp01.dbf'
  SIZE 50M AUTOEXTEND ON;
```

StudentDB — MySQL: Create Database & Users

Step-by-step: database creation, users, privileges

MySQL — Create Database & Users

```
-- 3. Create read-only reporting user
CREATE USER 'student_report'@'%'
  IDENTIFIED BY 'ReportOnly!2024';

GRANT SELECT ON StudentDB.*
  TO 'student_report'@'%';

-- 4. Flush privileges to apply changes
FLUSH PRIVILEGES;

-- 5. Verify
USE StudentDB;
SELECT user, host, password_expired
FROM mysql.user
WHERE user LIKE 'student%';

SHOW GRANTS FOR 'student_app'@'localhost';
```

Oracle — Tablespace & Schema User

```
-- 3. Create schema/owner user
CREATE USER student_schema
  IDENTIFIED BY "OracleStudents2024!"
  DEFAULT TABLESPACE students_ts
  TEMPORARY TABLESPACE students_temp
  QUOTA UNLIMITED ON students_ts;

-- 4. Grant system privileges
GRANT CREATE SESSION, CREATE TABLE,
  CREATE VIEW, CREATE SEQUENCE,
  CREATE PROCEDURE, CREATE TRIGGER,
  CREATE INDEX, CREATE SYNONYM
  TO student_schema;

-- 5. Verify
SELECT username, default_tablespace,
  account_status
FROM dba_users
WHERE username = 'STUDENT_SCHEMA';
```

StudentDB — MySQL: Core Table Definitions

departments, programs, instructors, students with all constraints

MySQL — departments

```
USE StudentDB;

CREATE TABLE departments (
  dept_id    INT NOT NULL AUTO_INCREMENT,
  dept_name  VARCHAR(100) NOT NULL,
  location   VARCHAR(100),
  budget     DECIMAL(15,2) DEFAULT 0.00,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_dept PRIMARY KEY (dept_id),
  CONSTRAINT uq_dept_name UNIQUE (dept_name)
) ENGINE=InnoDB;
```

MySQL — students

```
CREATE TABLE students (
  student_id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(50) NOT NULL,
  last_name  VARCHAR(50) NOT NULL,
  email      VARCHAR(100) NOT NULL,
  dob        DATE,
  phone      VARCHAR(20),
  prog_id    INT,
  gpa        DECIMAL(3,2) DEFAULT 0.00,
  status     ENUM('active','inactive',
    'graduated','suspended') DEFAULT 'active',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_student PRIMARY KEY (student_id),
  CONSTRAINT uq_student_email UNIQUE (email),
  CONSTRAINT fk_student_prog
    FOREIGN KEY (prog_id) REFERENCES
programs(prog_id),
  CONSTRAINT chk_gpa CHECK (gpa BETWEEN 0.00 AND 4.00)
) ENGINE=InnoDB;
```

StudentDB — MySQL: Core Table Definitions

departments, programs, instructors, students with all constraints

MySQL —programs

```
CREATE TABLE programs (
  prog_id          INT NOT NULL AUTO_INCREMENT,
  prog_name        VARCHAR(150) NOT NULL,
  dept_id          INT NOT NULL,
  credits_required INT DEFAULT 120,
  duration_years   TINYINT DEFAULT 4,
  CONSTRAINT pk_prog PRIMARY KEY (prog_id),
  CONSTRAINT fk_prog_dept
    FOREIGN KEY (dept_id) REFERENCES
departments(dept_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;
```

MySQL —courses

```
CREATE TABLE courses (
  course_id     INT NOT NULL AUTO_INCREMENT,
  course_name   VARCHAR(150) NOT NULL,
  prog_id       INT NOT NULL,
  instructor_id INT,
  credits       TINYINT DEFAULT 3,
  room          VARCHAR(20),
  CONSTRAINT pk_course PRIMARY KEY (course_id),
  CONSTRAINT fk_course_prog
    FOREIGN KEY (prog_id) REFERENCES
programs(prog_id),
  CONSTRAINT fk_course_instr
    FOREIGN KEY (instructor_id)
    REFERENCES instructors(instructor_id)
) ENGINE=InnoDB;
```

StudentDB — MySQL: Core Table Definitions

departments, programs, instructors, students with all constraints

MySQL — instructors and indices

```
CREATE TABLE instructors (
  instructor_id INT NOT NULL AUTO_INCREMENT,
  first_name    VARCHAR(50) NOT NULL,
  last_name     VARCHAR(50) NOT NULL,
  email         VARCHAR(100) NOT NULL,
  dept_id       INT NOT NULL,
  hire_date     DATE NOT NULL,
  salary        DECIMAL(10,2),
  CONSTRAINT pk_instr PRIMARY KEY (instructor_id),
  CONSTRAINT uq_instr_email UNIQUE (email),
  CONSTRAINT fk_instr_dept
    FOREIGN KEY (dept_id) REFERENCES
departments(dept_id)
) ENGINE=InnoDB;

CREATE INDEX idx_prog_dept ON programs(dept_id);
CREATE INDEX idx_instr_dept ON instructors(dept_id);
```

MySQL — enrollments PK, FK and constraints

```
-- M:N junction table
CREATE TABLE enrollments (
  student_id      INT NOT NULL,
  course_id       INT NOT NULL,
  semester        VARCHAR(10) NOT NULL,
  year            YEAR NOT NULL,
  enrollment_date DATE DEFAULT (CURRENT_DATE),
  status          VARCHAR(20) DEFAULT 'enrolled',
  CONSTRAINT pk_enroll
    PRIMARY KEY (student_id,course_id,semester,year),
  CONSTRAINT fk_enr_student
    FOREIGN KEY (student_id) REFERENCES
students(student_id),
  CONSTRAINT fk_enr_course
    FOREIGN KEY (course_id) REFERENCES
courses(course_id)
) ENGINE=InnoDB;
```

StudentDB — Oracle: Table Definitions

Oracle syntax with GENERATED IDENTITY, NUMBER types, tablespace assignment

Oracle — departments, programs, instructors

```
-- All tables go into students_ts tablespace
CREATE TABLE student_schema.departments (
  dept_id    NUMBER GENERATED ALWAYS AS IDENTITY,
  dept_name  VARCHAR2(100) NOT NULL,
  location   VARCHAR2(100),
  budget     NUMBER(15,2) DEFAULT 0,
  created_at DATE DEFAULT SYSDATE,
  CONSTRAINT pk_dept PRIMARY KEY (dept_id),
  CONSTRAINT uq_dept_name UNIQUE (dept_name)
) TABLESPACE students_ts;
```

Oracle — students, courses, enrollments

```
CREATE TABLE student_schema.students (
  student_id NUMBER GENERATED ALWAYS AS IDENTITY,
  first_name VARCHAR2(50) NOT NULL,
  last_name  VARCHAR2(50) NOT NULL,
  email      VARCHAR2(100) NOT NULL,
  dob        DATE,
  phone      VARCHAR2(20),
  prog_id    NUMBER,
  gpa        NUMBER(3,2) DEFAULT 0,
  status     VARCHAR2(20) DEFAULT 'active',
  created_at DATE DEFAULT SYSDATE,
  CONSTRAINT pk_student PRIMARY KEY (student_id),
  CONSTRAINT uq_student_email UNIQUE (email),
  CONSTRAINT fk_student_prog
    FOREIGN KEY (prog_id) REFERENCES programs(prog_id),
  CONSTRAINT chk_gpa CHECK (gpa BETWEEN 0 AND 4),
  CONSTRAINT chk_status CHECK (status IN
    ('active','inactive','graduated','suspended'))
) TABLESPACE students_ts;
```

StudentDB — Oracle: Table Definitions

Oracle syntax with GENERATED IDENTITY, NUMBER types, tablespace assignment

Oracle — departments, programs, instructors

```
CREATE TABLE student_schema.programs (
  prog_id          NUMBER GENERATED ALWAYS AS
IDENTITY,
  prog_name        VARCHAR2(150) NOT NULL,
  dept_id          NUMBER NOT NULL,
  credits_required NUMBER(3) DEFAULT 120,
  duration_years   NUMBER(1) DEFAULT 4,
  CONSTRAINT pk_prog PRIMARY KEY (prog_id),
  CONSTRAINT fk_prog_dept FOREIGN KEY (dept_id)
    REFERENCES departments(dept_id)
    ON DELETE SET NULL
) TABLESPACE students_ts;
```

Oracle — students, courses, enrollments

```
CREATE TABLE student_schema.courses (
  course_id     NUMBER GENERATED ALWAYS AS IDENTITY,
  course_name   VARCHAR2(150) NOT NULL,
  prog_id       NUMBER NOT NULL,
  instructor_id NUMBER,
  credits       NUMBER(1) DEFAULT 3,
  room          VARCHAR2(20),
  CONSTRAINT pk_course PRIMARY KEY (course_id),
  CONSTRAINT fk_course_prog
    FOREIGN KEY (prog_id) REFERENCES
programs(prog_id),
  CONSTRAINT fk_course_instr
    FOREIGN KEY (instructor_id)
    REFERENCES instructors(instructor_id)
) TABLESPACE students_ts;
```

StudentDB — Oracle: Table Definitions

Oracle syntax with GENERATED IDENTITY, NUMBER types, tablespace assignment

Oracle — departments, programs, instructors

```
CREATE TABLE student_schema.instructors (
  instructor_id NUMBER GENERATED ALWAYS AS IDENTITY,
  first_name    VARCHAR2(50) NOT NULL,
  last_name     VARCHAR2(50) NOT NULL,
  email         VARCHAR2(100) NOT NULL,
  dept_id       NUMBER NOT NULL,
  hire_date     DATE NOT NULL,
  salary        NUMBER(10,2),
  CONSTRAINT pk_instr PRIMARY KEY (instructor_id),
  CONSTRAINT uq_instr_email UNIQUE (email),
  CONSTRAINT fk_instr_dept FOREIGN KEY (dept_id)
    REFERENCES departments(dept_id)
) TABLESPACE students_ts;

-- Oracle requires explicit indexes on FK columns!
CREATE INDEX idx_prog_dept
  ON programs(dept_id) TABLESPACE students_ts;
CREATE INDEX idx_instr_dept
  ON instructors(dept_id) TABLESPACE students_ts;
```

Oracle — students, courses, enrollments

```
CREATE TABLE student_schema.enrollments (
  student_id      NUMBER NOT NULL,
  course_id       NUMBER NOT NULL,
  semester        VARCHAR2(10) NOT NULL,
  year_enrolled   NUMBER(4) NOT NULL,
  enrollment_date DATE DEFAULT SYSDATE,
  status          VARCHAR2(20) DEFAULT 'enrolled',
  CONSTRAINT pk_enroll
    PRIMARY
KEY(student_id,course_id,semester,year_enrolled),
  CONSTRAINT fk_enr_stu FOREIGN KEY (student_id)
    REFERENCES students(student_id),
  CONSTRAINT fk_enr_crs FOREIGN KEY (course_id)
    REFERENCES courses(course_id)
) TABLESPACE students_ts;
```

SECTION 04

AmazonShoppingDB — Full Build

E-Commerce · 10 Tables · Normalized · MySQL & Oracle DDL

AmazonShoppingDB — Normalized E-Commerce Schema

10 tables covering the full shopping lifecycle from product to payment

Table	PK	Key Columns	Purpose
categories	cat_id	cat_name, parent_cat_id(self-FK)	Hierarchical product categories
sellers	seller_id	company_name, email(UQ), rating, status	Marketplace sellers
products	product_id	product_name, cat_id(FK), seller_id(FK), price, sku(UQ)	Product catalog
customers	customer_id	first/last_name, email(UQ), loyalty_points, tier	Registered shoppers
addresses	address_id	customer_id(FK), street, city, state, zip, is_default	Shipping addresses
orders	order_id	customer_id(FK), ship_address_id(FK), order_date, status, total	Purchase orders
order_items	(order_id,product_id)	quantity, unit_price, discount, subtotal(computed)	M:N junction: orders↔products
payments	payment_id	order_id(FK), amount, method, status, transaction_id(UQ)	Payment records
reviews	review_id	product_id(FK), customer_id(FK), rating(1-5), review_text	Product reviews
inventory_log	log_id	product_id(FK), change_qty, reason, changed_by	Stock audit trail

AmazonShoppingDB — MySQL DDL Part 1

Create database, users, categories, sellers, products

MySQL — DB setup & categories/sellers

```
CREATE DATABASE IF NOT EXISTS AmazonShoppingDB
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

CREATE USER 'amazon_app'@'localhost'
  IDENTIFIED BY 'AmazonApp!2024';
GRANT ALL PRIVILEGES ON AmazonShoppingDB.*
  TO 'amazon_app'@'localhost';
CREATE USER 'amazon_ro'@'%'
  IDENTIFIED BY 'AmazonReport!2024';
GRANT SELECT ON AmazonShoppingDB.*
  TO 'amazon_ro'@'%';
FLUSH PRIVILEGES;
```

MySQL — products & customers tables

```
CREATE TABLE products (
  product_id   INT NOT NULL AUTO_INCREMENT,
  product_name VARCHAR(200) NOT NULL,
  cat_id       INT NOT NULL,
  seller_id    INT NOT NULL,
  price        DECIMAL(10,2) NOT NULL,
  stock_qty    INT DEFAULT 0,
  sku          VARCHAR(50) NOT NULL,
  description  TEXT,
  is_active    BOOLEAN DEFAULT TRUE,
  created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP
               ON UPDATE CURRENT_TIMESTAMP,
  CONSTRAINT pk_product  PRIMARY KEY (product_id),
  CONSTRAINT uq_sku      UNIQUE (sku),
  CONSTRAINT fk_prod_cat FOREIGN KEY (cat_id)
    REFERENCES categories(cat_id),
  CONSTRAINT fk_prod_seller FOREIGN KEY (seller_id)
    REFERENCES sellers(seller_id),
  CONSTRAINT chk_price CHECK (price >= 0),
  CONSTRAINT chk_stock CHECK (stock_qty >= 0)
) ENGINE=InnoDB;
```

AmazonShoppingDB — MySQL DDL Part 1

Create database, users, categories, sellers, products

MySQL — DB setup & categories/sellers

```
USE AmazonShoppingDB;

-- Self-referencing hierarchy
CREATE TABLE categories (
  cat_id        INT NOT NULL AUTO_INCREMENT,
  cat_name      VARCHAR(100) NOT NULL,
  parent_cat_id INT DEFAULT NULL,
  description   TEXT,
  CONSTRAINT pk_cat PRIMARY KEY (cat_id),
  CONSTRAINT fk_cat_parent
    FOREIGN KEY (parent_cat_id)
    REFERENCES categories(cat_id)
    ON DELETE SET NULL
) ENGINE=InnoDB;
```

MySQL — products & customers tables

```
CREATE TABLE customers (
  customer_id    INT NOT NULL AUTO_INCREMENT,
  first_name     VARCHAR(50) NOT NULL,
  last_name      VARCHAR(50) NOT NULL,
  email          VARCHAR(100) NOT NULL,
  phone          VARCHAR(20),
  loyalty_points INT DEFAULT 0,
  tier
ENUM('bronze','silver','gold','platinum')
                 DEFAULT 'bronze',
  created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_customer  PRIMARY KEY (customer_id),
  CONSTRAINT uq_cust_email UNIQUE (email)
) ENGINE=InnoDB;
```

AmazonShoppingDB — MySQL DDL Part 1

Create database, users, categories, sellers, products

MySQL — DB setup & categories/sellers

```
CREATE TABLE sellers (
  seller_id    INT NOT NULL AUTO_INCREMENT,
  company_name VARCHAR(150) NOT NULL,
  email        VARCHAR(100) NOT NULL,
  phone        VARCHAR(20),
  rating       DECIMAL(3,2) DEFAULT 5.00,
  join_date    DATE DEFAULT (CURRENT_DATE),
  status       ENUM('active','suspended','pending')
               DEFAULT 'pending',
  CONSTRAINT pk_seller  PRIMARY KEY (seller_id),
  CONSTRAINT uq_sel_email UNIQUE (email),
  CONSTRAINT chk_rating CHECK (rating BETWEEN 0 AND 5)
) ENGINE=InnoDB;
```

MySQL — products & customers tables

```
-- Indexes for frequent queries
CREATE INDEX idx_prod_cat    ON products(cat_id);
CREATE INDEX idx_prod_seller ON
products(seller_id);
CREATE INDEX idx_prod_price  ON products(price);
```

AmazonShoppingDB — MySQL DDL Part 2

orders, order_items, payments, reviews, inventory_log

MySQL — addresses, orders & order_items

```
CREATE TABLE addresses (
  address_id  INT NOT NULL AUTO_INCREMENT,
  customer_id INT NOT NULL,
  street      VARCHAR(200) NOT NULL,
  city        VARCHAR(100) NOT NULL,
  state       VARCHAR(50),
  zip         VARCHAR(20),
  country     VARCHAR(50) DEFAULT 'USA',
  is_default  BOOLEAN DEFAULT FALSE,
  CONSTRAINT pk_addr PRIMARY KEY (address_id),
  CONSTRAINT fk_addr_cust FOREIGN KEY (customer_id)
    REFERENCES customers(customer_id)
    ON DELETE CASCADE
) ENGINE=InnoDB;
```

MySQL — payments, reviews, inventory_log

```
CREATE TABLE payments (
  payment_id     INT NOT NULL AUTO_INCREMENT,
  order_id       INT NOT NULL,
  amount         DECIMAL(12,2) NOT NULL,
  method         ENUM('credit_card','debit_card',
    'paypal','amazon_pay','gift_card') NOT NULL,
  status         ENUM('pending','completed',
    'failed','refunded') DEFAULT 'pending',
  transaction_id VARCHAR(100),
  paid_at        TIMESTAMP NULL,
  CONSTRAINT pk_payment   PRIMARY KEY (payment_id),
  CONSTRAINT uq_txn_id    UNIQUE (transaction_id),
  CONSTRAINT fk_pay_order FOREIGN KEY (order_id)
    REFERENCES orders(order_id)
) ENGINE=InnoDB;
```

AmazonShoppingDB — MySQL DDL Part 2

orders, order_items, payments, reviews, inventory_log

MySQL — addresses, orders & order_items

```
CREATE TABLE orders (
  order_id        INT NOT NULL AUTO_INCREMENT,
  customer_id     INT NOT NULL,
  ship_address_id INT NOT NULL,
  order_date      TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status
ENUM('pending','processing','shipped',
    'delivered','cancelled','returned') DEFAULT
'pending',
  total_amount    DECIMAL(12,2) DEFAULT 0.00,
  notes           TEXT,
  CONSTRAINT pk_order PRIMARY KEY (order_id),
  CONSTRAINT fk_ord_cust FOREIGN KEY (customer_id)
    REFERENCES customers(customer_id),
  CONSTRAINT fk_ord_addr FOREIGN KEY (ship_address_id)
    REFERENCES addresses(address_id)
) ENGINE=InnoDB;
```

MySQL — payments, reviews, inventory_log

```
CREATE TABLE reviews (
  review_id     INT NOT NULL AUTO_INCREMENT,
  product_id    INT NOT NULL,
  customer_id   INT NOT NULL,
  rating        TINYINT NOT NULL,
  review_text   TEXT,
  helpful_votes INT DEFAULT 0,
  created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_review  PRIMARY KEY (review_id),
  CONSTRAINT uq_one_rev UNIQUE
(product_id,customer_id),
  CONSTRAINT fk_rev_prod FOREIGN KEY (product_id)
    REFERENCES products(product_id),
  CONSTRAINT fk_rev_cust FOREIGN KEY (customer_id)
    REFERENCES customers(customer_id),
  CONSTRAINT chk_rev_rating CHECK (rating BETWEEN 1
AND 5)
) ENGINE=InnoDB;
```

AmazonShoppingDB — MySQL DDL Part 2

orders, order_items, payments, reviews, inventory_log

MySQL — addresses, orders & order_items

```
-- M:N junction with computed subtotal
CREATE TABLE order_items (
  order_id   INT NOT NULL,
  product_id INT NOT NULL,
  quantity   INT NOT NULL DEFAULT 1,
  unit_price DECIMAL(10,2) NOT NULL,
  discount   DECIMAL(5,2) DEFAULT 0.00,
  subtotal   DECIMAL(12,2) GENERATED ALWAYS AS
    (quantity * unit_price * (1 - discount/100))
STORED,
  CONSTRAINT pk_oi PRIMARY KEY (order_id,product_id),
  CONSTRAINT fk_oi_order FOREIGN KEY (order_id)
    REFERENCES orders(order_id) ON DELETE CASCADE,
  CONSTRAINT fk_oi_prod  FOREIGN KEY (product_id)
    REFERENCES products(product_id)
) ENGINE=InnoDB;
```

MySQL — payments, reviews, inventory_log

```
-- Audit trail for all stock changes
CREATE TABLE inventory_log (
  log_id      INT NOT NULL AUTO_INCREMENT,
  product_id  INT NOT NULL,
  change_qty  INT NOT NULL,
  reason      VARCHAR(100),
  changed_by  VARCHAR(50),
  changed_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT pk_log      PRIMARY KEY (log_id),
  CONSTRAINT fk_log_prod FOREIGN KEY (product_id)
    REFERENCES products(product_id)
) ENGINE=InnoDB;

-- Verify all tables created
SHOW TABLES FROM AmazonShoppingDB;
SHOW CREATE TABLE order_items\G
```

SECTION 05

DDL Deep Dive

ALTER · DROP · TRUNCATE · RENAME · Views · Indexes · Sequences

DDL — ALTER TABLE Statements

Modify existing tables without dropping them — StudentDB & AmazonShoppingDB examples

MySQL — ALTER TABLE examples

```
USE StudentDB;

-- Add a new column
ALTER TABLE students
  ADD COLUMN middle_name VARCHAR(50)
  AFTER first_name;

-- Modify column data type / size
ALTER TABLE students
  MODIFY COLUMN phone VARCHAR(30);

ALTER TABLE student_grades RENAME TO grades;

USE AmazonShoppingDB;
-- Add discount column to products
ALTER TABLE products
  ADD COLUMN discount_pct DECIMAL(5,2) DEFAULT 0.00
  AFTER price;
```

Oracle — ALTER TABLE & DDL statements

```
-- Add column
ALTER TABLE student_schema.students
  ADD (middle_name VARCHAR2(50));

-- Modify column
ALTER TABLE student_schema.students
  MODIFY (phone VARCHAR2(30));

-- Rename column (Oracle 12c+)
ALTER TABLE student_schema.students
  RENAME COLUMN middle_name TO middle_initial;

-- Add constraint
ALTER TABLE student_schema.students
  ADD CONSTRAINT chk_email_fmt
  CHECK (email LIKE '%@%.%');
```

DDL — ALTER TABLE Statements

Modify existing tables without dropping them — StudentDB & AmazonShoppingDB examples

MySQL — ALTER TABLE examples

```
-- Add a new constraint
ALTER TABLE students
  ADD CONSTRAINT chk_status
  CHECK (status IN ('active','inactive',
    'graduated','suspended'));

-- Add a foreign key
ALTER TABLE grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;

-- Add an index
ALTER TABLE students
  ADD INDEX idx_last_name (last_name);
```

Oracle — ALTER TABLE & DDL statements

```
-- Add column
ALTER TABLE student_schema.students
  ADD (middle_name VARCHAR2(50));

-- Modify column
ALTER TABLE student_schema.students
  MODIFY (phone VARCHAR2(30));

-- Rename column
ALTER TABLE student_schema.students
  RENAME COLUMN middle_name TO middle_initial;

-- Add constraint
ALTER TABLE student_schema.students
  ADD CONSTRAINT chk_email_fmt
  CHECK (email LIKE '%@%.%');

-- Drop column
ALTER TABLE student_schema.students
  DROP COLUMN middle_initial;

-- Set column unused (faster on large tables)
ALTER TABLE student_schema.students
  SET UNUSED (phone);
-- Then drop unused later (during maintenance)
ALTER TABLE student_schema.students
  DROP UNUSED COLUMNS;

-- Rename table
ALTER TABLE student_schema.grades
  RENAME TO student_grades;

-- Add FK to existing table
ALTER TABLE student_schema.grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;
```

DDL — ALTER TABLE Statements

Modify existing tables without dropping them — StudentDB & AmazonShoppingDB examples

MySQL — ALTER TABLE examples

```
ALTER TABLE students
  ADD COLUMN middle_name VARCHAR(50)
  AFTER first_name;

-- Modify column data type / size
ALTER TABLE students
  MODIFY COLUMN phone VARCHAR(30);

-- Rename a column (MySQL 8.0+)
ALTER TABLE students
  RENAME COLUMN middle_name TO middle_initial;

-- Add a new constraint
ALTER TABLE students
  ADD CONSTRAINT chk_status
  CHECK (status IN ('active','inactive',
    'graduated','suspended'));

-- Add a foreign key
ALTER TABLE grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;

-- Add an index
ALTER TABLE students
  ADD INDEX idx_last_name (last_name);

-- Drop a column
ALTER TABLE students
  DROP COLUMN middle_initial;

-- Drop a constraint
ALTER TABLE students
  DROP INDEX idx_last_name;

-- Rename a table
ALTER TABLE grades
  RENAME TO student_grades;
```

Oracle — ALTER TABLE & DDL statements

```
-- Add column
ALTER TABLE student_schema.students
  ADD (middle_name VARCHAR2(50));

-- Modify column
ALTER TABLE student_schema.students
  MODIFY (phone VARCHAR2(30));

-- Rename column
ALTER TABLE student_schema.students
  RENAME COLUMN middle_name TO middle_initial;

-- Add constraint
ALTER TABLE student_schema.students
  ADD CONSTRAINT chk_email_fmt
  CHECK (email LIKE '%@%.%');

-- Drop column
ALTER TABLE student_schema.students
  DROP COLUMN middle_initial;

-- Set column unused (faster on large tables)
ALTER TABLE student_schema.students
  SET UNUSED (phone);
-- Then drop unused later (during maintenance)
ALTER TABLE student_schema.students
  DROP UNUSED COLUMNS;

-- Rename table
ALTER TABLE student_schema.grades
  RENAME TO student_grades;

-- Add FK to existing table
ALTER TABLE student_schema.grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;
```

DDL — ALTER TABLE Statements

Modify existing tables without dropping them — StudentDB & AmazonShoppingDB examples

MySQL — ALTER TABLE examples

```
ALTER TABLE students
  ADD COLUMN middle_name VARCHAR(50)
  AFTER first_name;

-- Modify column data type / size
ALTER TABLE students
  MODIFY COLUMN phone VARCHAR(30);

-- Rename a column (MySQL 8.0+)
ALTER TABLE students
  RENAME COLUMN middle_name TO middle_initial;

-- Add a new constraint
ALTER TABLE students
  ADD CONSTRAINT chk_status
  CHECK (status IN ('active','inactive',
    'graduated','suspended'));

-- Add a foreign key
ALTER TABLE grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;

-- Add an index
ALTER TABLE students
  ADD INDEX idx_last_name (last_name);

-- Drop a column
ALTER TABLE students
  DROP COLUMN middle_initial;

-- Drop a constraint
ALTER TABLE students
  DROP INDEX idx_last_name;

-- Rename a table
ALTER TABLE grades
  RENAME TO student_grades;
```

Oracle — ALTER TABLE & DDL statements

```
-- Add column
ALTER TABLE student_schema.students
  ADD (middle_name VARCHAR2(50));

-- Modify column
ALTER TABLE student_schema.students
  MODIFY (phone VARCHAR2(30));

-- Rename column
ALTER TABLE student_schema.students
  RENAME COLUMN middle_name TO middle_initial;

-- Add constraint
ALTER TABLE student_schema.students
  ADD CONSTRAINT chk_email_fmt
  CHECK (email LIKE '%@%.%');

-- Drop column
ALTER TABLE student_schema.students
  DROP COLUMN middle_initial;

-- Set column unused (faster on large tables)
ALTER TABLE student_schema.students
  SET UNUSED (phone);
-- Then drop unused later (during maintenance)
ALTER TABLE student_schema.students
  DROP UNUSED COLUMNS;

-- Rename table
ALTER TABLE student_schema.grades
  RENAME TO student_grades;

-- Add FK to existing table
ALTER TABLE student_schema.grades
  ADD CONSTRAINT fk_grade_student
  FOREIGN KEY (student_id)
  REFERENCES students(student_id)
  ON DELETE CASCADE;
```

DDL — Views, Indexes & Sequences

CREATE VIEW · CREATE INDEX · CREATE SEQUENCE with StudentDB examples

MySQL — Views & Indexes

```
USE StudentDB;

-- Simple view: student with program name
CREATE OR REPLACE VIEW vw_student_details AS
  SELECT s.student_id,
    CONCAT(s.first_name,' ',s.last_name) AS full_name,
    s.email, s.gpa, s.status,
    p.prog_name, d.dept_name
  FROM students s
  JOIN programs p ON s.prog_id = p.prog_id
  JOIN departments d ON p.dept_id = d.dept_id;

-- Updatable view example
CREATE OR REPLACE VIEW vw_active_students AS
  SELECT student_id, first_name, last_name,
    email, gpa FROM students
  WHERE status = 'active'
  WITH CHECK OPTION;

-- Query the view (same as a table)
SELECT * FROM vw_student_details
WHERE dept_name = 'Computer Science'
ORDER BY gpa DESC;

-- Composite index for GPA + status queries
CREATE INDEX idx_gpa_status
  ON students(status, gpa DESC);

-- Full-text index for search
ALTER TABLE products
  ADD FULLTEXT INDEX ft_prod_name(product_name,description);

-- Use fulltext index
SELECT * FROM products
WHERE MATCH(product_name,description)
  AGAINST('laptop gaming' IN BOOLEAN MODE);
```

Oracle — Views, Sequences & Indexes

```
-- Materialized view (caches result on disk)
CREATE MATERIALIZED VIEW
  student_schema.mv_enrollment_summary
  BUILD IMMEDIATE
  REFRESH COMPLETE ON DEMAND AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    COUNT(e.course_id) AS courses_enrolled,
    AVG(g.grade_points) AS avg_gpa
  FROM students s
  LEFT JOIN enrollments e USING(student_id)
  LEFT JOIN grades g USING(student_id)
  GROUP BY s.student_id, s.first_name, s.last_name;

-- Force refresh
EXEC DBMS_MVIEW.REFRESH('mv_enrollment_summary');

-- Regular view
CREATE OR REPLACE VIEW
  student_schema.vw_student_details AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    s.email, s.gpa, p.prog_name
  FROM students s JOIN programs p
    ON s.prog_id = p.prog_id;

-- Sequence for legacy tables
CREATE SEQUENCE student_schema.payment_seq
  START WITH 1000 INCREMENT BY 1
  NOCACHE NOCYCLE NOORDER;

-- Use in INSERT
INSERT INTO payments VALUES
  (payment_seq.NEXTVAL, ...);

-- Function-based index
CREATE INDEX idx_upper_email
  ON students(UPPER(email));
```

DDL — Views, Indexes & Sequences

CREATE VIEW · CREATE INDEX · CREATE SEQUENCE with StudentDB examples

MySQL — Views & Indexes

```
USE StudentDB;

-- Simple view: student with program name
CREATE OR REPLACE VIEW vw_student_details AS
  SELECT s.student_id,
    CONCAT(s.first_name,' ',s.last_name) AS full_name,
    s.email, s.gpa, s.status,
    p.prog_name, d.dept_name
  FROM students s
  JOIN programs p ON s.prog_id = p.prog_id
  JOIN departments d ON p.dept_id = d.dept_id;

-- Updatable view example
CREATE OR REPLACE VIEW vw_active_students AS
  SELECT student_id, first_name, last_name,
    email, gpa FROM students
  WHERE status = 'active'
  WITH CHECK OPTION;

-- Query the view (same as a table)
SELECT * FROM vw_student_details
WHERE dept_name = 'Computer Science'
ORDER BY gpa DESC;

-- Composite index for GPA + status queries
CREATE INDEX idx_gpa_status
  ON students(status, gpa DESC);

-- Full-text index for search
ALTER TABLE products
  ADD FULLTEXT INDEX ft_prod_name(product_name,description);

-- Use fulltext index
SELECT * FROM products
WHERE MATCH(product_name,description)
  AGAINST('laptop gaming' IN BOOLEAN MODE);
```

Oracle — Views, Sequences & Indexes

```
-- Materialized view (caches result on disk)
CREATE MATERIALIZED VIEW
  student_schema.mv_enrollment_summary
  BUILD IMMEDIATE
  REFRESH COMPLETE ON DEMAND AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    COUNT(e.course_id) AS courses_enrolled,
    AVG(g.grade_points) AS avg_gpa
  FROM students s
  LEFT JOIN enrollments e USING(student_id)
  LEFT JOIN grades g USING(student_id)
  GROUP BY s.student_id, s.first_name, s.last_name;

-- Force refresh
EXEC DBMS_MVIEW.REFRESH('mv_enrollment_summary');

-- Regular view
CREATE OR REPLACE VIEW
  student_schema.vw_student_details AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    s.email, s.gpa, p.prog_name
  FROM students s JOIN programs p
    ON s.prog_id = p.prog_id;

-- Sequence for legacy tables
CREATE SEQUENCE student_schema.payment_seq
  START WITH 1000 INCREMENT BY 1
  NOCACHE NOCYCLE NOORDER;

-- Use in INSERT
INSERT INTO payments VALUES
  (payment_seq.NEXTVAL, ...);

-- Function-based index
CREATE INDEX idx_upper_email
  ON students(UPPER(email));
```

DDL — Views, Indexes & Sequences

CREATE VIEW · CREATE INDEX · CREATE SEQUENCE with StudentDB examples

MySQL — Views & Indexes

```
USE StudentDB;

-- Simple view: student with program name
CREATE OR REPLACE VIEW vw_student_details AS
  SELECT s.student_id,
    CONCAT(s.first_name,' ',s.last_name) AS full_name,
    s.email, s.gpa, s.status,
    p.prog_name, d.dept_name
  FROM students s
  JOIN programs p ON s.prog_id = p.prog_id
  JOIN departments d ON p.dept_id = d.dept_id;

-- Updatable view example
CREATE OR REPLACE VIEW vw_active_students AS
  SELECT student_id, first_name, last_name,
    email, gpa FROM students
  WHERE status = 'active'
  WITH CHECK OPTION;

-- Query the view (same as a table)
SELECT * FROM vw_student_details
WHERE dept_name = 'Computer Science'
ORDER BY gpa DESC;
```

Oracle — Views, Sequences & Indexes

```
-- Materialized view (caches result on disk)
CREATE MATERIALIZED VIEW
  student_schema.mv_enrollment_summary
  BUILD IMMEDIATE
  REFRESH COMPLETE ON DEMAND AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    COUNT(e.course_id) AS courses_enrolled,
    AVG(g.grade_points) AS avg_gpa
  FROM students s
  LEFT JOIN enrollments e USING(student_id)
  LEFT JOIN grades g USING(student_id)
  GROUP BY s.student_id, s.first_name, s.last_name;

-- Force refresh
EXEC DBMS_MVIEW.REFRESH('mv_enrollment_summary');

-- Regular view
CREATE OR REPLACE VIEW
  student_schema.vw_student_details AS
  SELECT s.student_id,
    s.first_name||' '||s.last_name AS full_name,
    s.email, s.gpa, p.prog_name
  FROM students s JOIN programs p
    ON s.prog_id = p.prog_id;

-- Sequence for legacy tables
CREATE SEQUENCE student_schema.payment_seq
  START WITH 1000 INCREMENT BY 1
  NOCACHE NOCYCLE NOORDER;
```

DDL — Views, Indexes & Sequences

CREATE VIEW · CREATE INDEX · CREATE SEQUENCE with StudentDB examples

MySQL — Views & Indexes

```
-- Composite index for GPA + status queries
CREATE INDEX idx_gpa_status
  ON students(status, gpa DESC);

-- Full-text index for search
ALTER TABLE products
  ADD FULLTEXT INDEX
ft_prod_name(product_name,description);
```

Oracle — Views, Sequences & Indexes

```
  (payment_seq.NEXTVAL, ...);

-- Function-based index
CREATE INDEX idx_upper_email
  ON students(UPPER(email));

-- Bitmap index (for low-cardinality columns)
CREATE BITMAP INDEX idx_status_bmp
  ON students(status);

-- Invisible index (test before enabling)
CREATE INDEX idx_gpa
  ON students(gpa) INVISIBLE;
ALTER INDEX idx_gpa VISIBLE;
```

DDL — Views, Indexes & Sequences

CREATE VIEW · CREATE INDEX · CREATE SEQUENCE with StudentDB examples

MySQL — Views & Indexes

```
-- Use fulltext index
SELECT * FROM products
WHERE MATCH(product_name,description)
  AGAINST('laptop gaming' IN BOOLEAN MODE);

-- Covering index (includes all needed columns)
CREATE INDEX idx_order_cust_status
  ON orders(customer_id, status, order_date);

-- DROP a view
DROP VIEW IF EXISTS vw_student_details;
```

Oracle — Views, Sequences & Indexes

```
-- Function-based index
CREATE INDEX idx_upper_email
  ON students(UPPER(email));

-- Bitmap index (for low-cardinality columns)
CREATE BITMAP INDEX idx_status_bmp
  ON students(status);

-- Invisible index (test before enabling)
CREATE INDEX idx_gpa
  ON students(gpa) INVISIBLE;
ALTER INDEX idx_gpa VISIBLE;
```

SECTION 06

DML Deep Dive

INSERT · UPDATE · DELETE · SELECT · JOINs · Subqueries · Real Data

DML — INSERT: Populating StudentDB & AmazonShoppingDB

Single row, multi-row, and INSERT from SELECT examples

MySQL — INSERT: StudentDB data

```
USE StudentDB;

-- Single row INSERT
INSERT INTO departments (dept_name, location, budget)
VALUES ('Computer Science', 'Building A', 500000.00);

INSERT INTO departments (dept_name, location, budget)
VALUES ('Business Administration', 'Building B',
300000.00),
       ('Mathematics', 'Building C', 200000.00),
       ('Engineering', 'Building D', 800000.00);
```

MySQL — INSERT: AmazonShoppingDB data

```
USE AmazonShoppingDB;

-- Categories (with self-reference)
INSERT INTO categories (cat_name,parent_cat_id)
VALUES ('Electronics', NULL),
       ('Clothing', NULL),
       ('Books', NULL),
       ('Laptops', 1),        -- child of
Electronics
       ('Smartphones', 1),
       ('Men Clothing', 2),
       ('Women Clothing', 2);
```

DML — SELECT: JOINs, Subqueries & Aggregations

The most powerful SQL skill — retrieving data intelligently

MySQL — SELECT with JOINs: StudentDB

```
USE StudentDB;

-- INNER JOIN: students with their program & dept
SELECT s.student_id,
  CONCAT(s.first_name,' ',s.last_name) AS
student_name,
  s.email, s.gpa, p.prog_name, d.dept_name
FROM students s
INNER JOIN programs p ON s.prog_id = p.prog_id
INNER JOIN departments d ON p.dept_id = d.dept_id
WHERE s.status = 'active'
ORDER BY s.gpa DESC;
```

MySQL — SELECT: AmazonShoppingDB queries

```
USE AmazonShoppingDB;

-- Customer order history with totals
SELECT c.first_name, c.last_name, c.email,
  COUNT(o.order_id) AS total_orders,
  SUM(o.total_amount) AS lifetime_value,
  MAX(o.order_date) AS last_order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
ORDER BY lifetime_value DESC
LIMIT 10;
```

DML — SELECT: JOINs, Subqueries & Aggregations

The most powerful SQL skill — retrieving data intelligently

MySQL — SELECT with JOINs: StudentDB

```
-- LEFT JOIN: all students even without enrollments
SELECT s.student_id, s.first_name,
  COUNT(e.course_id) AS courses_enrolled
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
  AND e.semester = 'FALL' AND e.year = 2024
GROUP BY s.student_id, s.first_name
ORDER BY courses_enrolled DESC;

-- Subquery: students with GPA above average
SELECT student_id, first_name, last_name, gpa
FROM students
WHERE gpa > (SELECT AVG(gpa) FROM students
             WHERE status='active')
ORDER BY gpa DESC;
```

MySQL — SELECT: AmazonShoppingDB queries

```
-- Order details with product names
SELECT o.order_id, o.order_date,
  CONCAT(c.first_name,' ',c.last_name) AS customer,
  p.product_name, oi.quantity,
  oi.unit_price, oi.discount, oi.subtotal
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
WHERE o.status = 'delivered'
  AND o.order_date >= DATE_SUB(NOW(), INTERVAL 30 DAY)
ORDER BY o.order_date DESC;
```

DML — SELECT: JOINs, Subqueries & Aggregations

The most powerful SQL skill — retrieving data intelligently

MySQL — SELECT with JOINs: StudentDB

```
-- LEFT JOIN: all students even without enrollments
SELECT s.student_id, s.first_name,
  COUNT(e.course_id) AS courses_enrolled
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
  AND e.semester = 'FALL' AND e.year = 2024
GROUP BY s.student_id, s.first_name
ORDER BY courses_enrolled DESC;

-- Subquery: students with GPA above average
SELECT student_id, first_name, last_name, gpa
FROM students
WHERE gpa > (SELECT AVG(gpa) FROM students
             WHERE status='active')
ORDER BY gpa DESC;
```

MySQL — SELECT: AmazonShoppingDB queries

```
-- Top selling products by revenue
SELECT p.product_name, cat.cat_name,
  SUM(oi.quantity) AS units_sold,
  SUM(oi.subtotal) AS total_revenue,
  AVG(r.rating) AS avg_rating
FROM products p
JOIN categories cat ON p.cat_id = cat.cat_id
JOIN order_items oi ON p.product_id = oi.product_id
LEFT JOIN reviews r ON p.product_id = r.product_id
GROUP BY p.product_id, p.product_name, cat.cat_name
ORDER BY total_revenue DESC
LIMIT 10;

-- Window function: rank customers by spending
SELECT customer_id, first_name,
  SUM(total_amount) AS total_spent,
  RANK() OVER (ORDER BY SUM(total_amount) DESC)
    AS spending_rank
FROM customers c
JOIN orders o USING(customer_id)
GROUP BY customer_id;
```

DML — SELECT: JOINs, Subqueries & Aggregations

The most powerful SQL skill — retrieving data intelligently

MySQL — SELECT with JOINs: StudentDB

```
-- Correlated subquery: each student's best grade
SELECT s.first_name, c.course_name, g.grade_letter
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE g.grade_points = (
  SELECT MAX(g2.grade_points)
  FROM grades g2
  WHERE g2.student_id = s.student_id
);

-- Aggregate: GPA stats by department
SELECT d.dept_name,
  COUNT(s.student_id) AS student_count,
  ROUND(AVG(s.gpa),2) AS avg_gpa,
  MAX(s.gpa) AS highest_gpa,
  MIN(s.gpa) AS lowest_gpa
FROM departments d
JOIN programs p ON d.dept_id = p.dept_id
JOIN students s ON p.prog_id = s.prog_id
WHERE s.status = 'active'
GROUP BY d.dept_id, d.dept_name
HAVING AVG(s.gpa) > 3.0
ORDER BY avg_gpa DESC;
```

MySQL — SELECT: AmazonShoppingDB queries

```
-- Top selling products by revenue
SELECT p.product_name, cat.cat_name,
  SUM(oi.quantity) AS units_sold,
  SUM(oi.subtotal) AS total_revenue,
  AVG(r.rating) AS avg_rating
FROM products p
JOIN categories cat ON p.cat_id = cat.cat_id
JOIN order_items oi ON p.product_id = oi.product_id
LEFT JOIN reviews r ON p.product_id = r.product_id
GROUP BY p.product_id, p.product_name, cat.cat_name
ORDER BY total_revenue DESC
LIMIT 10;

-- Window function: rank customers by spending
SELECT customer_id, first_name,
  SUM(total_amount) AS total_spent,
  RANK() OVER (ORDER BY SUM(total_amount) DESC)
    AS spending_rank
FROM customers c
JOIN orders o USING(customer_id)
GROUP BY customer_id;
```

DML — UPDATE & DELETE with Real Examples

Modify and remove records safely — always use WHERE!

MySQL — UPDATE statements

```
-- Update with JOIN (update from another table)
UPDATE students s
JOIN enrollments e ON s.student_id = e.student_id
SET s.gpa = ROUND(
  (SELECT AVG(g.grade_points)
   FROM grades g
   WHERE g.student_id = s.student_id), 2)
WHERE e.year = 2024;

-- Update with CASE (conditional update)
UPDATE customers
SET tier = CASE
  WHEN loyalty_points >= 20000 THEN 'platinum'
  WHEN loyalty_points >= 8000  THEN 'gold'
  WHEN loyalty_points >= 2000  THEN 'silver'
  ELSE 'bronze'
END;
```

MySQL — DELETE & Oracle UPDATE/DELETE

```
-- Delete old records (with LIMIT for safety)
DELETE FROM inventory_log
WHERE changed_at < DATE_SUB(NOW(),INTERVAL 1 YEAR)
LIMIT 1000;

-- TRUNCATE vs DELETE
TRUNCATE TABLE inventory_log;  -- fast, no log
-- vs
DELETE FROM inventory_log;     -- slower, logged

-- Transaction-safe delete
START TRANSACTION;
DELETE FROM grades WHERE student_id = 10;
DELETE FROM enrollments WHERE student_id = 10;
DELETE FROM payments WHERE student_id = 10;
DELETE FROM students WHERE student_id = 10;
COMMIT;  -- or ROLLBACK if something fails
```

DML — UPDATE & DELETE with Real Examples

Modify and remove records safely — always use WHERE!

MySQL — UPDATE statements

```
-- Update order total from order_items
UPDATE orders o
SET total_amount = (
  SELECT SUM(subtotal)
  FROM order_items oi
  WHERE oi.order_id = o.order_id
);

-- Safe update mode bypass for batch updates
SET SQL_SAFE_UPDATES = 0;
UPDATE products
SET is_active = FALSE
WHERE stock_qty = 0;
SET SQL_SAFE_UPDATES = 1;
```

MySQL — DELETE & Oracle UPDATE/DELETE

```
-- ── Oracle UPDATE & DELETE ──────────────────
UPDATE student_schema.students
SET gpa = ROUND(
  (SELECT AVG(g.grade_points)
   FROM grades g
   WHERE g.student_id = students.student_id), 2)
WHERE EXISTS (SELECT 1 FROM grades g
  WHERE g.student_id = students.student_id);

-- MERGE (UPSERT) — Oracle & MySQL 8+
MERGE INTO students s
USING (SELECT 1001 AS id, 3.85 AS new_gpa
       FROM DUAL) src
ON (s.student_id = src.id)
WHEN MATCHED THEN
  UPDATE SET s.gpa = src.new_gpa
WHEN NOT MATCHED THEN
  INSERT (student_id,gpa) VALUES (src.id,src.new_gpa);
```

SECTION 07

DCL — Data Control Language

GRANT · REVOKE · Roles · Profiles · Row-Level Security

DCL — GRANT & REVOKE: Full Examples

Control exactly who can do what — for StudentDB & AmazonShoppingDB

MySQL — GRANT & REVOKE

```
-- ── StudentDB Permissions ──────────────────
-- App user: full CRUD on StudentDB
GRANT SELECT, INSERT, UPDATE, DELETE
  ON StudentDB.* TO 'student_app'@'localhost';

-- Report user: read-only
GRANT SELECT ON StudentDB.*
  TO 'student_report'@'%';

-- Specific column-level permission
GRANT SELECT (student_id, first_name,
  last_name, gpa)
  ON StudentDB.students
  TO 'student_report'@'%';
```

Oracle — GRANT, REVOKE & Row-Level Security

```
-- ── StudentDB Oracle Grants ────────────────
GRANT SELECT, INSERT, UPDATE, DELETE
  ON student_schema.students
  TO student_app_user;

-- Grant execute on stored procedure
GRANT EXECUTE ON student_schema.sp_enroll_student
  TO student_app_user;

-- Object-level grants with GRANT OPTION
GRANT SELECT ON student_schema.vw_student_details
  TO report_user WITH GRANT OPTION;
```

DCL — GRANT & REVOKE: Full Examples

Control exactly who can do what — for StudentDB & AmazonShoppingDB

MySQL — GRANT & REVOKE

```
-- Grant with GRANT OPTION (can re-grant)
GRANT ALL ON StudentDB.*
  TO 'student_admin'@'localhost'
  WITH GRANT OPTION;

-- Create and grant a role (MySQL 8+)
CREATE ROLE 'student_readonly';
GRANT SELECT ON StudentDB.*
  TO 'student_readonly';
GRANT 'student_readonly' TO 'student_report'@'%';
SET DEFAULT ROLE ALL TO 'student_report'@'%';

-- ── AmazonShoppingDB Permissions ──────────
GRANT SELECT, INSERT, UPDATE
  ON AmazonShoppingDB.products
  TO 'amazon_app'@'localhost';
```

Oracle — GRANT, REVOKE & Row-Level Security

```
-- Create role
CREATE ROLE student_reader;
GRANT SELECT ON student_schema.students
  TO student_reader;
GRANT SELECT ON student_schema.courses
  TO student_reader;
-- Assign role to user
GRANT student_reader TO report_user;

-- Column-level grant
GRANT SELECT (student_id, first_name,
  last_name, gpa)
  ON student_schema.students TO report_user;

-- REVOKE
REVOKE INSERT ON student_schema.students
  FROM student_app_user;
REVOKE student_reader FROM report_user;
```

DCL — GRANT & REVOKE: Full Examples

Control exactly who can do what — for StudentDB & AmazonShoppingDB

MySQL — GRANT & REVOKE

```
GRANT SELECT ON AmazonShoppingDB.orders
  TO 'amazon_app'@'localhost';

-- REVOKE specific permission
REVOKE DELETE ON StudentDB.students
  FROM 'student_app'@'localhost';

-- REVOKE all
REVOKE ALL PRIVILEGES ON StudentDB.*
  FROM 'student_report'@'%';

-- View grants
SHOW GRANTS FOR 'student_app'@'localhost';

-- Flush to apply
FLUSH PRIVILEGES;
```

Oracle — GRANT, REVOKE & Row-Level Security

```
-- Row-Level Security using VPD (Virtual Private DB)
-- Policy: students can only see their own grades
CREATE OR REPLACE FUNCTION grade_policy(
  schema_name VARCHAR2, table_name VARCHAR2)
  RETURN VARCHAR2 AS
BEGIN
  RETURN 'student_id = SYS_CONTEXT(''USERENV'',
    ''SESSION_USER_ID'')';
END;
/
DBMS_RLS.ADD_POLICY(
  object_schema => 'student_schema',
  object_name   => 'grades',
  policy_name   => 'grade_student_policy',
  function_schema => 'sys',
  policy_function => 'grade_policy',
  statement_types => 'SELECT');
```

Advanced DDL — Stored Procedures & Triggers

Automate business logic inside the database

MySQL — Stored Procedure & Trigger

```
DELIMITER //
-- Stored Procedure: Enroll a student
CREATE PROCEDURE sp_enroll_student(
  IN p_student_id INT,
  IN p_course_id  INT,
  IN p_semester   VARCHAR(10),
  IN p_year       YEAR,
  OUT p_result    VARCHAR(100)
)
BEGIN
  DECLARE v_count INT;
  -- Check if already enrolled
  SELECT COUNT(*) INTO v_count
  FROM enrollments
  WHERE student_id=p_student_id
    AND course_id=p_course_id
    AND semester=p_semester
    AND year=p_year;

  IF v_count > 0 THEN
    SET p_result = 'Already enrolled';
  ELSE
    INSERT INTO enrollments
      (student_id,course_id,semester,year)
    VALUES (p_student_id,p_course_id,p_semester,p_year);
    SET p_result = 'Enrolled successfully';
  END IF;
END //
```

Oracle — Procedure, Function & Trigger

```
-- Stored Procedure (Oracle PL/SQL)
CREATE OR REPLACE PROCEDURE
  student_schema.sp_update_gpa(p_student_id NUMBER)
AS
  v_avg_gpa NUMBER(3,2);
BEGIN
  SELECT ROUND(AVG(grade_points),2)
  INTO v_avg_gpa
  FROM student_schema.grades
  WHERE student_id = p_student_id;

  UPDATE student_schema.students
  SET gpa = NVL(v_avg_gpa, 0)
  WHERE student_id = p_student_id;

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('GPA updated: '||v_avg_gpa);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No grades found');
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;
END sp_update_gpa;
/
```

Advanced DDL — Stored Procedures & Triggers

Automate business logic inside the database

MySQL — Stored Procedure & Trigger

```
DELIMITER //
-- Stored Procedure: Enroll a student
CREATE PROCEDURE sp_enroll_student(
  IN p_student_id INT,
  IN p_course_id  INT,
  IN p_semester   VARCHAR(10),
  IN p_year       YEAR,
  OUT p_result    VARCHAR(100)
)
BEGIN
  DECLARE v_count INT;
  -- Check if already enrolled
  SELECT COUNT(*) INTO v_count
  FROM enrollments
  WHERE student_id=p_student_id
    AND course_id=p_course_id
    AND semester=p_semester
    AND year=p_year;

  IF v_count > 0 THEN
    SET p_result = 'Already enrolled';
  ELSE
    INSERT INTO enrollments
      (student_id,course_id,semester,year)
    VALUES (p_student_id,p_course_id,p_semester,p_year);
    SET p_result = 'Enrolled successfully';
  END IF;
END //
```

Oracle — Procedure, Function & Trigger

```
-- Function: calculate student tier
CREATE OR REPLACE FUNCTION
  amazon_schema.get_customer_tier(p_points NUMBER)
  RETURN VARCHAR2 AS
BEGIN
  IF    p_points >= 20000 THEN RETURN 'platinum';
  ELSIF p_points >= 8000  THEN RETURN 'gold';
  ELSIF p_points >= 2000  THEN RETURN 'silver';
  ELSE  RETURN 'bronze';
  END IF;
END;
/
```

Advanced DDL — Stored Procedures & Triggers

Automate business logic inside the database

MySQL — Stored Procedure & Trigger

```
-- AFTER INSERT trigger: log inventory changes
CREATE TRIGGER trg_stock_decrease
AFTER UPDATE ON products
FOR EACH ROW
BEGIN
  IF NEW.stock_qty < OLD.stock_qty THEN
    INSERT INTO inventory_log
      (product_id, change_qty, reason, changed_by)
    VALUES (NEW.product_id,
      NEW.stock_qty - OLD.stock_qty,
      'Sale/order', USER());
  END IF;
END //

DELIMITER ;

-- Call stored procedure
CALL sp_enroll_student(1, 2, 'FALL', 2024, @result);
SELECT @result;
```

Oracle — Procedure, Function & Trigger

```
-- BEFORE INSERT trigger
CREATE OR REPLACE TRIGGER
  student_schema.trg_student_email_lower
BEFORE INSERT OR UPDATE ON students
FOR EACH ROW
BEGIN
  :NEW.email := LOWER(:NEW.email);
END;
/

-- Execute procedure
EXEC student_schema.sp_update_gpa(1001);
```

SECTION 08

Backup & Recovery

mysqldump · MySQL Export/Import · Oracle RMAN · Data Pump

MySQL Backup — mysqldump & mysqlpump

Command-line backup and restore for StudentDB & AmazonShoppingDB

mysqldump — Backup Commands

```
# ── Full database backup ──────────────────────
mysqldump -u root -p StudentDB >
/backups/StudentDB_$(date +%Y%m%d).sql

# ── Backup with all options (production) ─────
mysqldump -u root -p   --single-transaction   --
routines   --triggers   --events   --add-drop-table
--comments   StudentDB > /backups/StudentDB_full.sql

# ── Backup multiple databases ─────────────────
mysqldump -u root -p   --databases StudentDB
AmazonShoppingDB   > /backups/both_dbs.sql

# ── Backup all databases ──────────────────────
mysqldump -u root -p --all-databases   --single-
transaction > /backups/all_dbs.sql
```

MySQL Restore & mysqlpump

```
# ── Restore from backup ───────────────────────
mysql -u root -p StudentDB <   /backups/StudentDB_full.sql

# ── Restore from compressed backup ────────────
gunzip -c /backups/StudentDB_20240115.sql.gz |   mysql -u
root -p StudentDB

# ── Restore to different database ─────────────
# First create target DB
mysql -u root -p -e   "CREATE DATABASE StudentDB_restore;"
# Then restore
mysql -u root -p StudentDB_restore <
/backups/StudentDB_full.sql

# ── Point-in-time recovery using binlog ───────
# Enable binary logging in my.cnf:
# [mysqld]
# log_bin = /var/log/mysql/mysql-bin.log
# server-id = 1
```

MySQL Backup — mysqldump & mysqlpump

Command-line backup and restore for StudentDB & AmazonShoppingDB

mysqldump — Backup Commands

```
# ── Backup specific tables only ──────────────
mysqldump -u root -p StudentDB   students enrollments
grades   > /backups/students_tables.sql

# ── Backup schema only (no data) ──────────────
mysqldump -u root -p --no-data StudentDB   >
/backups/StudentDB_schema.sql

# ── Backup data only (no schema) ──────────────
mysqldump -u root -p --no-create-info StudentDB   >
/backups/StudentDB_data.sql

# ── Compressed backup ─────────────────────────
mysqldump -u root -p StudentDB |   gzip >
/backups/StudentDB_$(date +%Y%m%d).sql.gz
```

MySQL Restore & mysqlpump

```
# Show binary log position
SHOW MASTER STATUS;
SHOW BINARY LOGS;

# Apply binlog from specific position
mysqlbinlog   --start-datetime="2024-01-15 10:00:00"
--stop-datetime="2024-01-15 11:00:00"
/var/log/mysql/mysql-bin.000001 |   mysql -u root -p

# ── mysqlpump (parallel, faster) ──────────────
mysqlpump -u root -p   --parallel-schemas=4:StudentDB
--default-parallelism=2   StudentDB >
/backups/StudentDB_pump.sql

# ── MySQL Shell logical dump (8.0+) ───────────
# mysqlsh root@localhost --sql
# util.dumpSchemas(['StudentDB'],'/backups/dump/')
# util.loadDump('/backups/dump/')
```

Oracle Backup — RMAN & Data Pump

Recovery Manager (RMAN) and Data Pump exp/imp for StudentDB

Oracle RMAN — Backup & Recovery

```
# ── Connect to RMAN ──────────────────────────
rman target /
# or
rman target sys/password@ORCL

# ── Configure RMAN settings ──────────────────
RMAN> CONFIGURE RETENTION POLICY TO
        RECOVERY WINDOW OF 7 DAYS;
RMAN> CONFIGURE BACKUP OPTIMIZATION ON;
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO DISK;
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK
        FORMAT '/rman_backups/%d_%T_%s.bkp';

# ── Full database backup ─────────────────────
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;

# ── Incremental backup (level 0 = full base) ─
RMAN> BACKUP INCREMENTAL LEVEL 0 DATABASE;
# Level 1 = changes since last level 0
RMAN> BACKUP INCREMENTAL LEVEL 1 DATABASE;
```

Oracle Data Pump — expdp & impdp

```
# ── Create directory object ──────────────────
sqlplus / as sysdba
SQL> CREATE DIRECTORY dp_dir AS '/dp_backups';
SQL> GRANT READ, WRITE ON DIRECTORY dp_dir
       TO student_schema;

# ── Full schema export (expdp) ───────────────
expdp student_schema/password@ORCL
SCHEMAS=student_schema   DIRECTORY=dp_dir
DUMPFILE=students_%date%.dmp
LOGFILE=students_exp.log   COMPRESSION=ALL

# ── Export specific tables ───────────────────
expdp student_schema/password@ORCL
TABLES=students,enrollments,grades   DIRECTORY=dp_dir
DUMPFILE=students_tables.dmp   LOGFILE=tables_exp.log
```

Oracle Backup — RMAN & Data Pump

Recovery Manager (RMAN) and Data Pump exp/imp for StudentDB

Oracle RMAN — Backup & Recovery

```
# ── Backup specific tablespace ──────────────
RMAN> BACKUP TABLESPACE students_ts;

# ── Backup control file and spfile ──────────
RMAN> BACKUP CURRENT CONTROLFILE;
RMAN> BACKUP SPFILE;

# ── List all backups ────────────────────────
RMAN> LIST BACKUP SUMMARY;
RMAN> LIST BACKUP OF DATABASE;

# ── Restore and recover ─────────────────────
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;

# ── Point-in-time recovery ──────────────────
RMAN> RUN {
  SET UNTIL TIME "TO_DATE('2024-01-15 10:00',
    'YYYY-MM-DD HH24:MI')";
  RESTORE DATABASE;
  RECOVER DATABASE;
  ALTER DATABASE OPEN RESETLOGS;
}
```

Oracle Data Pump — expdp & impdp

```
# ── Export with query filter ────────────────
expdp student_schema/password@ORCL   TABLES=students
QUERY=students:'"WHERE status=''active''"'
DIRECTORY=dp_dir   DUMPFILE=active_students.dmp

# ── Full schema import (impdp) ──────────────
impdp student_schema/password@ORCL
SCHEMAS=student_schema   DIRECTORY=dp_dir
DUMPFILE=students_20240115.dmp
LOGFILE=students_imp.log

# ── Import to different schema ──────────────
impdp system/password@ORCL
REMAP_SCHEMA=student_schema:student_test
REMAP_TABLESPACE=students_ts:test_ts   DIRECTORY=dp_dir
DUMPFILE=students_20240115.dmp

# ── Network import (direct DB-to-DB) ────────
impdp system/password@TARGET
NETWORK_LINK=source_db_link   SCHEMAS=student_schema
LOGFILE=network_imp.log
```

SECTION 09

Python · R · Java · C++ APIs

Real code examples for connecting and performing CRUD on both databases

Python API — mysql-connector, cx_Oracle, SQLAlchemy

The most popular data science language connecting to both databases

Python — MySQL StudentDB CRUD

```
import mysql.connector
import pandas as pd
from sqlalchemy import create_engine

# ── Direct connection ──────────────────────
conn = mysql.connector.connect(
    host='localhost', port=3306,
    user='student_app', password='StrongPass!2024',
    database='StudentDB',
    autocommit=False
)
cursor = conn.cursor(dictionary=True)
```

Python — Oracle cx_Oracle & AmazonDB

```
import cx_Oracle
import pandas as pd
from sqlalchemy import create_engine

# ── Oracle connection ──────────────────────
dsn = cx_Oracle.makedsn('localhost', 1521,
                        service_name='ORCL')
conn = cx_Oracle.connect(
    user='student_schema',
    password='OracleStudents2024!',
    dsn=dsn, encoding='UTF-8')
cursor = conn.cursor()

# INSERT with bind variables
cursor.execute("""
    INSERT INTO students
      (first_name,last_name,email,prog_id,gpa)
    VALUES (:1,:2,:3,:4,:5)
""", ('Khalid','Rahimi','k.rahimi@uni.edu',1,3.8))
conn.commit()
```

Python API — mysql-connector, cx_Oracle, SQLAlchemy

The most popular data science language connecting to both databases

Python — MySQL StudentDB CRUD

```
# INSERT
cursor.execute("""
    INSERT INTO students
      (first_name,last_name,email,prog_id,gpa)
    VALUES (%s,%s,%s,%s,%s)
""", ('Fatima','Ahmadi','f.ahmadi@uni.edu',1,3.9))
conn.commit()
new_id = cursor.lastrowid
print(f"Inserted student ID: {new_id}")
```

Python — Oracle cx_Oracle & AmazonDB

```
# SELECT with named binds
cursor.execute("""
    SELECT student_id, first_name||' '||last_name
      AS full_name, gpa
    FROM students
    WHERE gpa > :min_gpa
    ORDER BY gpa DESC
    FETCH FIRST 10 ROWS ONLY
""", {'min_gpa': 3.5})

for row in cursor:
    print(f"ID:{row[0]} {row[1]} GPA:{row[2]}")
```

Python API — mysql-connector, cx_Oracle, SQLAlchemy

The most popular data science language connecting to both databases

Python — MySQL StudentDB CRUD

```
# SELECT with parameterized query
cursor.execute("""
    SELECT s.student_id,
      CONCAT(s.first_name,' ',s.last_name) AS name,
      s.gpa, p.prog_name
    FROM students s
    JOIN programs p ON s.prog_id=p.prog_id
    WHERE s.gpa > %s AND s.status = %s
    ORDER BY s.gpa DESC
""", (3.5, 'active'))
rows = cursor.fetchall()
for row in rows:
    print(f"{row['name']} - GPA: {row['gpa']}")
```

Python — Oracle cx_Oracle & AmazonDB

```
# ── Bulk insert with executemany ──────────────
products_data = [
    ('iPad Pro 12.9', 4, 1, 1099.99, 100, 'APPL-IPAD-
PRO'),
    ('AirPods Pro',   5, 1, 249.99,  500, 'APPL-
AIRPODS'),
]
cursor.executemany("""
    INSERT INTO amazon_schema.products
      (product_name,cat_id,seller_id,
       price,stock_qty,sku)
    VALUES (:1,:2,:3,:4,:5,:6)
""", products_data)
conn.commit()
```

Python API — mysql-connector, cx_Oracle, SQLAlchemy

The most popular data science language connecting to both databases

Python — MySQL StudentDB CRUD

```
# ── pandas + SQLAlchemy ──────────────────────
engine = create_engine(
    "mysql+mysqlconnector://student_app:"
    "StrongPass!2024@localhost/StudentDB")

df = pd.read_sql("""
    SELECT s.*, p.prog_name, d.dept_name
    FROM students s
    JOIN programs p USING(prog_id)
    JOIN departments d USING(dept_id)
""", con=engine)
print(df.describe())
print(df.groupby('dept_name')['gpa'].mean())

cursor.close(); conn.close()
```

Python — Oracle cx_Oracle & AmazonDB

```
# ── SQLAlchemy for Oracle ────────────────────
engine = create_engine(
    'oracle+cx_oracle://student_schema:'
    'OracleStudents2024!@localhost:1521/'
    '?service_name=ORCL')

df = pd.read_sql('SELECT * FROM students', engine)
print(df.head())
cursor.close(); conn.close()
```

R Language & Java JDBC — Database Connections

Statistical analysis with R · Enterprise Java JDBC connections

R Language — DBI, RMySQL, dplyr

```r
# Install packages (run once)
# install.packages(c("DBI","RMySQL","dplyr","ggplot2"))

library(DBI)
library(RMySQL)
library(dplyr)
library(ggplot2)

# ── Connect to StudentDB ──────────────────────
con <- dbConnect(RMySQL::MySQL(),
  dbname   = "StudentDB",
  host     = "localhost",
  port     = 3306,
  user     = "student_app",
  password = "StrongPass!2024"
)

# ── Simple query ──────────────────────────────
students_df <- dbGetQuery(con,
  "SELECT s.*, p.prog_name, d.dept_name
   FROM students s
   JOIN programs p USING(prog_id)
   JOIN departments d USING(dept_id)
   WHERE s.status = 'active'")

print(head(students_df))
print(summary(students_df$gpa))
```

Java JDBC — StudentDB & AmazonShoppingDB

```java
import java.sql.*;
import java.util.*;

public class StudentDBManager {
  private static final String MYSQL_URL =
    "jdbc:mysql://localhost:3306/StudentDB"
    + "?useSSL=false&serverTimezone=UTC";
  private static final String ORACLE_URL =
    "jdbc:oracle:thin:@localhost:1521:ORCL";

  // ── MySQL Connection ──────────────────────
  public Connection getMySQLConnection()
      throws SQLException {
    return DriverManager.getConnection(
      MYSQL_URL, "student_app", "StrongPass!2024");
  }

  // ── Oracle Connection ─────────────────────
  public Connection getOracleConnection()
      throws SQLException {
    return DriverManager.getConnection(
      ORACLE_URL,
      "student_schema", "OracleStudents2024!");
  }
```

R Language & Java JDBC — Database Connections

Statistical analysis with R · Enterprise Java JDBC connections

R Language — DBI, RMySQL, dplyr

```r
# ── dplyr lazy evaluation (generates SQL) ──────
students_tbl <- tbl(con, "students")
active_students <- students_tbl %>%
  filter(status == "active", gpa > 3.0) %>%
  select(student_id, first_name, gpa) %>%
  arrange(desc(gpa)) %>%
  collect()  # executes SQL query here
```

Java JDBC — StudentDB & AmazonShoppingDB

```java
// ── SELECT with PreparedStatement ──────────
public List<Map<String,Object>> getTopStudents(
    double minGpa) throws SQLException {
  List<Map<String,Object>> results = new ArrayList<>();
  String sql = "SELECT s.student_id, " +
    "CONCAT(s.first_name,' ',s.last_name) AS name,"+
    "s.gpa, p.prog_name " +
    "FROM students s JOIN programs p " +
    "ON s.prog_id=p.prog_id " +
    "WHERE s.gpa > ? AND s.status='active' " +
    "ORDER BY s.gpa DESC LIMIT 10";
  try (Connection conn = getMySQLConnection();
       PreparedStatement ps =
         conn.prepareStatement(sql)) {
    ps.setDouble(1, minGpa);
    ResultSet rs = ps.executeQuery();
    ResultSetMetaData md = rs.getMetaData();
    while (rs.next()) {
      Map<String,Object> row = new HashMap<>();
      for (int i=1;i<=md.getColumnCount();i++)
        row.put(md.getColumnName(i),rs.getObject(i));
      results.add(row);
    }
  }
  return results;
}
```

R Language & Java JDBC — Database Connections

Statistical analysis with R · Enterprise Java JDBC connections

R Language — DBI, RMySQL, dplyr

```r
# ── GPA distribution plot ──────────────────────
ggplot(students_df, aes(x=gpa, fill=dept_name)) +
  geom_histogram(bins=20, alpha=0.7) +
  labs(title="GPA Distribution by Department",
       x="GPA", y="Count") +
  theme_minimal()

# ── INSERT from R ──────────────────────────────
dbExecute(con,
  "INSERT INTO grades (student_id,course_id,
    grade_letter,grade_points,semester)
   VALUES (?, ?, ?, ?, ?)",
  list(1, 2, 'A', 4.0, 'FALL'))

dbDisconnect(con)
```

Java JDBC — StudentDB & AmazonShoppingDB

```java
  // ── INSERT with transaction ──────────────
  public void enrollStudent(int studentId,
      int courseId, String semester, int year)
      throws SQLException {
    String sql = "INSERT INTO enrollments " +
      "(student_id,course_id,semester,year) " +
      "VALUES (?,?,?,?)";
    try (Connection conn = getMySQLConnection()) {
      conn.setAutoCommit(false);
      try (PreparedStatement ps =
              conn.prepareStatement(sql)) {
        ps.setInt(1, studentId);
        ps.setInt(2, courseId);
        ps.setString(3, semester);
        ps.setInt(4, year);
        ps.executeUpdate();
        conn.commit();
      } catch (Exception e) {
        conn.rollback(); throw e;
      }
    }
  }
}
```

C++ API — libmysqlclient & Oracle OCCI

Low-level, high-performance database access from C++

C++ MySQL Connector/C

```
// Compile: g++ -o dbapp dbapp.cpp \
//   -lmysqlclient -I/usr/include/mysql

#include <mysql/mysql.h>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
  MYSQL *conn = mysql_init(NULL);
  if (!conn) {
    cerr << "MySQL init failed" << endl;
    return 1;
  }

  // ── Connect ──────────────────────────
  if (!mysql_real_connect(conn,
        "localhost",     // host
        "student_app",   // user
        "StrongPass!2024",// password
        "StudentDB",     // database
        3306,            // port
        NULL, 0)) {
    cerr << "Connect error: "
         << mysql_error(conn) << endl;
    mysql_close(conn);
    return 1;
  }
```

C++ Oracle OCCI & ODBC

```
// Oracle OCCI (C++)
// Compile: g++ -o oracle_app oracle.cpp \
//   -locci -lclntsh

#include <occi.h>
#include <iostream>
using namespace oracle::occi;
using namespace std;

int main() {
  Environment *env =
Environment::createEnvironment();
  Connection *conn = nullptr;
  try {
    conn = env->createConnection(
      "student_schema",          // user
      "OracleStudents2024!",     // password
      "localhost:1521/ORCL");    // connect
string

    cout << "Connected to Oracle!" << endl;
```

C++ API — libmysqlclient & Oracle OCCI

Low-level, high-performance database access from C++

C++ MySQL Connector/C

```
cout << "Connected to StudentDB!" << endl;
  mysql_set_character_set(conn, "utf8mb4");

  // ── SELECT with prepared statement ──────
  MYSQL_STMT *stmt = mysql_stmt_init(conn);
  const char *query =
    "SELECT student_id, first_name, gpa "
    "FROM students WHERE gpa > ? "
    "ORDER BY gpa DESC LIMIT 5";
  mysql_stmt_prepare(stmt, query, strlen(query));

  double min_gpa = 3.5;
  MYSQL_BIND bind_in[1] = {};
  bind_in[0].buffer_type = MYSQL_TYPE_DOUBLE;
  bind_in[0].buffer = &min_gpa;
  mysql_stmt_bind_param(stmt, bind_in);
```

C++ Oracle OCCI & ODBC

```
// ── SELECT with Statement ──────────────
Statement *stmt = conn->createStatement(
  "SELECT student_id, first_name, gpa "
  "FROM students WHERE gpa > :1 "
  "ORDER BY gpa DESC FETCH FIRST 5 ROWS ONLY");
stmt->setDouble(1, 3.5);

ResultSet *rs = stmt->executeQuery();
while (rs->next()) {
  cout << rs->getInt(1) << " "
       << rs->getString(2) << " GPA: "
       << rs->getDouble(3) << endl;
}
stmt->closeResultSet(rs);
```

C++ API — libmysqlclient & Oracle OCCI

Low-level, high-performance database access from C++

C++ MySQL Connector/C

```
// Result buffers
int sid; char fname[51]; double gpa;
unsigned long fname_len;
MYSQL_BIND bind_out[3] = {};
bind_out[0].buffer_type=MYSQL_TYPE_LONG;
bind_out[0].buffer=&sid;
bind_out[1].buffer_type=MYSQL_TYPE_STRING;
bind_out[1].buffer=fname;
bind_out[1].buffer_length=51;
bind_out[1].length=&fname_len;
bind_out[2].buffer_type=MYSQL_TYPE_DOUBLE;
bind_out[2].buffer=&gpa;
```

C++ Oracle OCCI & ODBC

```
    // ── INSERT ──────────────────────────
    Statement *ins = conn->createStatement(
      "INSERT INTO students "
      "(first_name,last_name,email,prog_id,gpa) "
      "VALUES (:1,:2,:3,:4,:5)");
    ins->setString(1, "Ahmed");
    ins->setString(2, "Hassan");
    ins->setString(3, "ahmed.h@uni.edu");
    ins->setInt(4, 1);
    ins->setDouble(5, 3.75);
    ins->executeUpdate();
    conn->commit();
    conn->terminateStatement(ins);
    cout << "Student inserted!" << endl;

  } catch (SQLException &e) {
    cerr << "Oracle error: "
         << e.getMessage() << endl;
    if (conn) { conn->rollback(); }
  }
  if (conn) env->terminateConnection(conn);
  Environment::terminateEnvironment(env);
  return 0;
}
```

C++ API — libmysqlclient & Oracle OCCI

Low-level, high-performance database access from C++

C++ MySQL Connector/C

```
  mysql_stmt_bind_result(stmt, bind_out);
  mysql_stmt_execute(stmt);
  while (!mysql_stmt_fetch(stmt))
    cout<<sid<<" "<<fname<<" GPA:"<<gpa<<endl;

  mysql_stmt_close(stmt);
  mysql_close(conn);
  return 0;
}
```

C++ Oracle OCCI & ODBC

```
/* ── ODBC (cross-database) ──────────────────
#include <sql.h>
#include <sqlext.h>
// Works with MySQL, Oracle, MSSQL, etc.
// Configure DSN in /etc/odbc.ini:
// [StudentDB-MySQL]
// Driver=MySQL ODBC 8.0
// Server=localhost
// Database=StudentDB
// User=student_app
// Password=StrongPass!2024
*/
```

🎓 Thank You!

You are now ready to build, manage & query

professional databases!

🌐 **MyWebUniversity.com**

▶ **youtube.com/@MyWebUniversity**

▶ **youtube.com/@MyWebUniversity-Dari**

💼 Free Linux · Unix · Windows · Programming Tutorials

👍 **LIKE** · 🔔 **SUBSCRIBE** · 🔗 **SHARE** · 💬 **COMMENT**

Don't forget to SUBSCRIBE for more free SQL, Linux & programming tutorials!