Dos and Don'ts on Database Creation and SQL Development: Best Practices

1. Design and Planning

Dos

- Do define the purpose and scope clearly before designing the database to understand what data needs to be stored and how it will be used.
- Do analyze and gather all required data and business rules thoroughly.
- Do create a conceptual data model first (entities, relationships), then logical (tables, columns), and finally physical design (indexes, partitions). Using diagrams helps visualize structure and relationships.
- Do use normalization rules (up to 3NF or higher) to avoid data redundancy and ensure integrity (e.g., no repeated data, dependencies are correct).
- Do choose appropriate data types that match data requirements, favoring storage efficiency and validity (e.g., use date/time data types for dates).
- Do choose meaningful, consistent, and clear naming conventions for tables, columns, indexes, constraints (e.g., snake_case or CamelCase consistently). This aids readability and maintenance.
- Do specify primary keys and unique constraints to enforce entity integrity.
- Do define foreign keys to maintain referential integrity between related tables; use cascading rules cautiously.
- Do document schema, columns, relationships, constraints, and intended use clearly for future developers and users.
- Do consider future scalability and flexibility — anticipate growth in data size, users, and evolving requirements.
- Do use indexing wisely on frequently filtered or joined columns to boost query performance.
- Do optimize for common query patterns by analyzing workload and tuning accordingly (e.g., adding composite indexes if needed).
- Do use views, stored procedures, and functions to encapsulate complex logic and improve security.

Don'ts

- Don't ignore normalization; avoid storing redundant or duplicated data. Avoid multi-valued columns or storing lists in one column.
- Don't use ambiguous or inconsistent naming conventions. Avoid abbreviations that are unclear and reserved SQL keywords.
- Don't create wide tables with too many columns; break down logically related data into separate tables.
- Don't store calculated or derived data directly if it can be computed dynamically unless for performance reasons (denormalization).
- Don't omit constraints and validation at the database level; relying only on application logic risks data integrity.
- Don't over-index. Excessive indexes slow down writes and increase maintenance overhead.
- Don't use inappropriate data types (e.g., store dates as strings) as it hinders validation and query efficiency.
- Don't ignore NULL semantics; know when to allow NULL versus requiring values to avoid ambiguous data.

2. Development Best Practices

Dos

- Do write clear, maintainable SQL code. Use indentation, line breaks, and comments.
- Do use parameterized queries or prepared statements to guard against SQL injection vulnerabilities.
- Do write modular code—use views, stored procedures, and functions to encapsulate logic and promote reusability.
- Do test your queries against realistic data sets to verify correctness, performance, and edge cases.
- Do use transactions to ensure atomicity, consistency, isolation, and durability (ACID).
- Do use backup and recovery strategies regularly in development and production.
- Do monitor query performance and execution plans; optimize indexes and queries as needed.
- Do handle errors gracefully using proper exception handling and logging.
- Do maintain version control of database schema and SQL scripts to track changes and enable rollback.
- Do use database migration tools for schema changes instead of manual ad-hoc scripts wherever possible.

Don'ts

- Don't hardcode values in queries; use parameters instead for flexibility and security.
- Don't fetch more data than needed; use projections (select only needed columns) and pagination to limit data volume.
- Don't use SELECT * unless necessary. It harms performance and can break applications upon schema changes.
- Don't rely solely on application code for enforcing data integrity. The database should enforce constraints and validations.
- Don't ignore transaction boundaries; incomplete or long transactions risk locking and inconsistencies.
- Don't neglect indexing maintenance; improper or outdated indexes degrade performance.
- Don't postpone database documentation; lack of documentation increases technical debt and maintenance difficulty.

3. Performance and Scalability

Dos

- Do analyze query plans (`EXPLAIN`, `EXPLAIN ANALYZE`, etc.) for performance insights.
- Do index columns used in JOINs, WHERE clauses, and ORDER BY predicates.
- Do consider denormalization strategically if queries involve complex joins and performance matters.
- Do use partitioning for very large tables to manage data and improve query efficiency.
- Do implement caching layers when possible to reduce database load.
- Do design for horizontal or vertical scaling depending on expected data growth.
- Do archive or purge obsolete data to keep database lean and performant.

Don'ts

- Don't index every column without analysis; index use comes at write and storage cost.
- Don't ignore locks and concurrency control; poorly designed queries lead to deadlocks or performance bottlenecks.

- Don't assume that denormalization is always bad; use it thoughtfully for performance improvement with due caution.
- Don't design the schema without considering query patterns; schema should support efficient data retrieval for expected use cases.

4. Security and Compliance

Dos

- Do implement access control: least privilege principle for database users and roles.
- Do encrypt sensitive data both at rest and in transit (database-level or disk-level encryption).
- Do audit and log critical database activities.
- Do comply with relevant regulations (e.g., GDPR, HIPAA) regarding data privacy and protection.
- Do use secure connections (TLS/SSL) for database access.

Don'ts

- Don't expose database credentials or connection strings in source code or public repositories.
- Don't grant overly permissive privileges to users or applications.
- Don't ignore patches and updates for the database server and software stack.

Useful Summary Table of Best Practices

Aspect	**Dos**	**Don'ts**
Design	Normalize data, use PK/FK, consistent names, document	Duplicate data, inconsistent naming, wide tables

Development	Modular SQL, parameterized queries, transactions, testing	SELECT *, hardcoding, ignore integrity, long transactions
Performance	Use indexes wisely, analyze plans, partition large tables	Over-index, ignore locks, neglect scaling considerations
Security	Least privilege, encrypt data, audit, comply regulations	Expose credentials, grant excessive rights, ignore patches