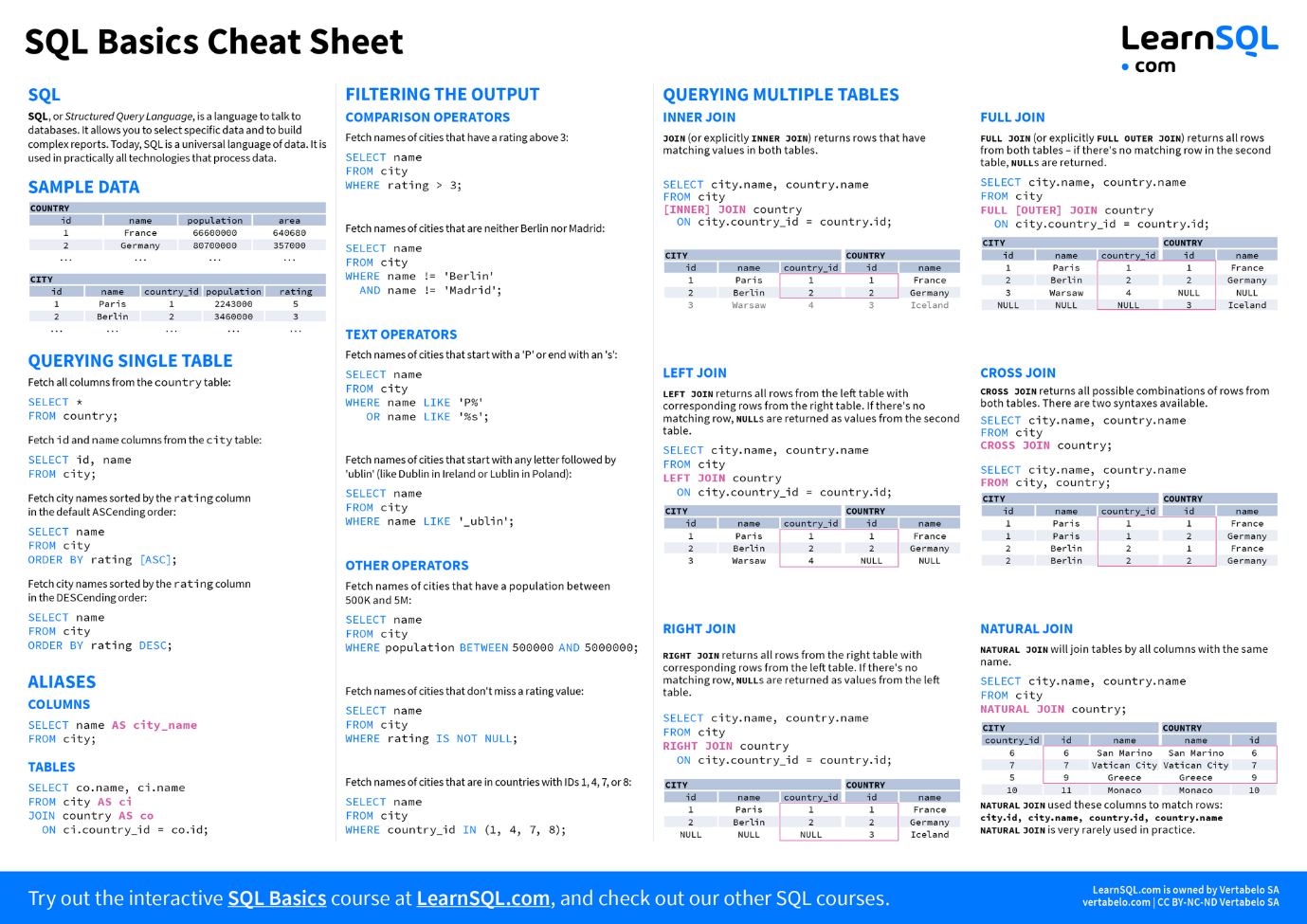
Coding



|  |  |  |
| --- | --- | --- |
| 1 | Querying | select \* from a |
| 2 | Sorting | order by b |
| 3 | Aliasing | Columns, Tables |
| 4 | Joins | a join b on a.id = b.id |
| 5 | Filtering | where a > 10, DISTINCT |
| 6 | Text operations | like ‘%000’ |
| 7 | Other operations | between, null, is not null, in, LIMIT, TOP |
| 8 | Types of Joins | Inner, left, right, full outer, cross, natural |
| 9 | Aggregations | Group by, having, Count (\*), Sum, AVG, Min, Max, |
| 10 | Type cast, Replace |  |
| 11 | Subqueries | Nested queries, correlated queries |
| 12 | Set operations | UNION, INTERSECT, EXCEPT |
| 13 | Window functions | ROW\_NUMBER, RANK, DENSE\_RANK |
| 14 | Data Modelling | Normalization, Indexing |
| 15 | DML | Insert into a values b, Update set, Delete from |

Questions

**What are ACID principles?**

**Atomicity**: all parts of a transaction are completed successfully/ the entire transaction is rolled back

**Consistency**: from one valid state to another

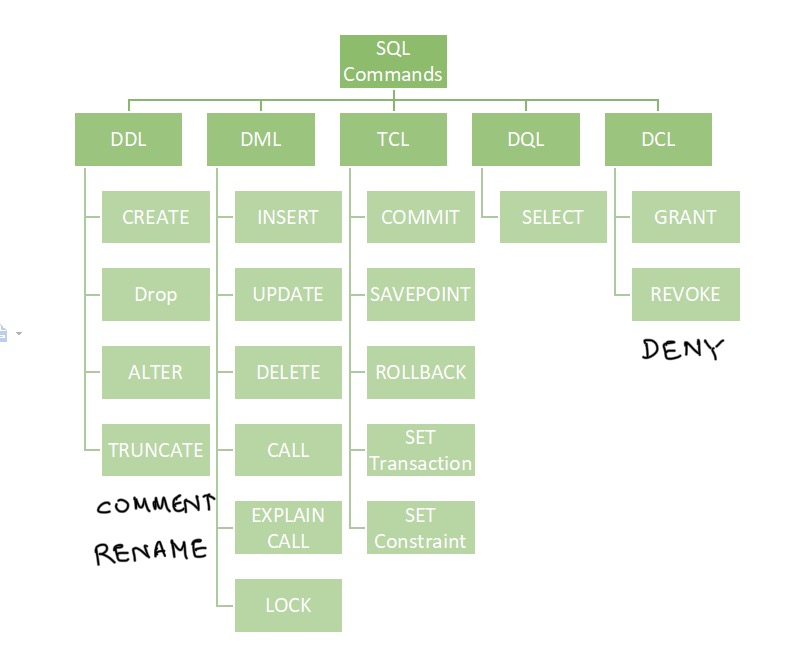
For example, if there are integrity rules like foreign key constraints, check constraints, data types, or uniqueness, any transaction that violates these rules is not allowed to commit.

**Isolation**: Ensures that transactions are executed independently of one another.

Leaves system from one consistent state to another

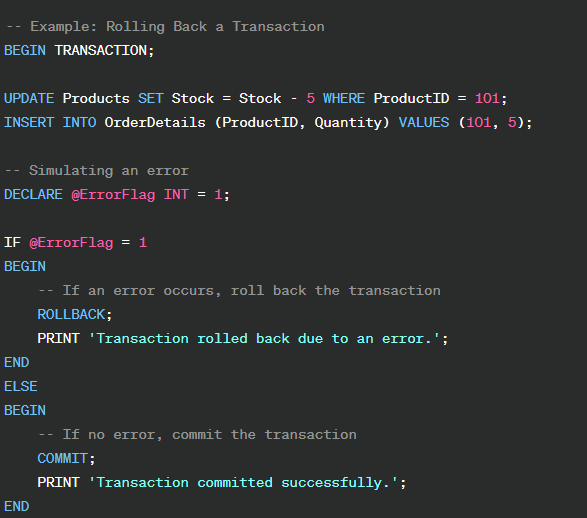
**Durability**: Ensures that once a transaction is committed, the changes are permanent, even in the case of a system failure. The data is reliably stored and can be recovered.

**What are the types of SQL commands?**



|  |  |
| --- | --- |
| **Stored proc vs functions** | <https://youtube.com/shorts/mQ18DJ0O_vg?si=NRQJGfKtTsLxLbjy>  <https://youtube.com/shorts/Bc99o3be4sU?si=fjMa23r3obr_Lkol>  <https://youtube.com/shorts/JZKgQ4ALYWQ?si=97pds2T-WUE4HY9Y> |
| **Views** | <https://youtube.com/shorts/KbEQ8FdF6Qs?si=Nt0oq4NwLj4BNny8> |
| **CTEs**  Use subqueries over CTEs for large data | <https://youtube.com/shorts/LJhNaOii9yk?si=ZhJEWX5GWpscNBup>  <https://youtube.com/shorts/8-rppJ-YOLc?si=-NG5Ltnco3njgi0z>  <https://youtube.com/shorts/fX1iv0EWZW0?si=uxuKzPFQ6TkAMwYr> |
| **Temp tables** | <https://youtube.com/shorts/CVj6Rne0T3M?si=aE4VsqKGcY_nfy99> |

**Give example of TCL type of SQL code**



**Give definitions of various objects constituting Data Base structures**

1. **Tables**: Fundamental units, rows, and columns.
2. **Columns**: Attribute representation with specific data types.
3. **Rows**: Contain actual data entries for unique instances.
4. **Primary Key:** Unique identifier ensuring data integrity.
5. **Foreign Key:** Links columns between tables, establishes relationships.
6. **Indexes**: Data structures for efficient data retrieval.
7. **Relationships**: Define how tables are related (one-to-one, one-to-many).
8. **Data Types**: Specify the kind of data in columns for accuracy.
9. **Constraints**: Rules and conditions for data (primary key, unique, check).
10. **Views**: Virtual tables generated by queries for simplicity.
11. **Stored Procedures and Functions**: Precompiled sets of SQL statements for logic and rule

**Example Schema:**

* **Customers** **Table**:
  + CustomerID (Primary Key)
  + Name
  + Email
* **Orders** **Table**:
  + OrderID (Primary Key)
  + OrderDate
  + CustomerID (Foreign Key, references CustomerID in Customers)

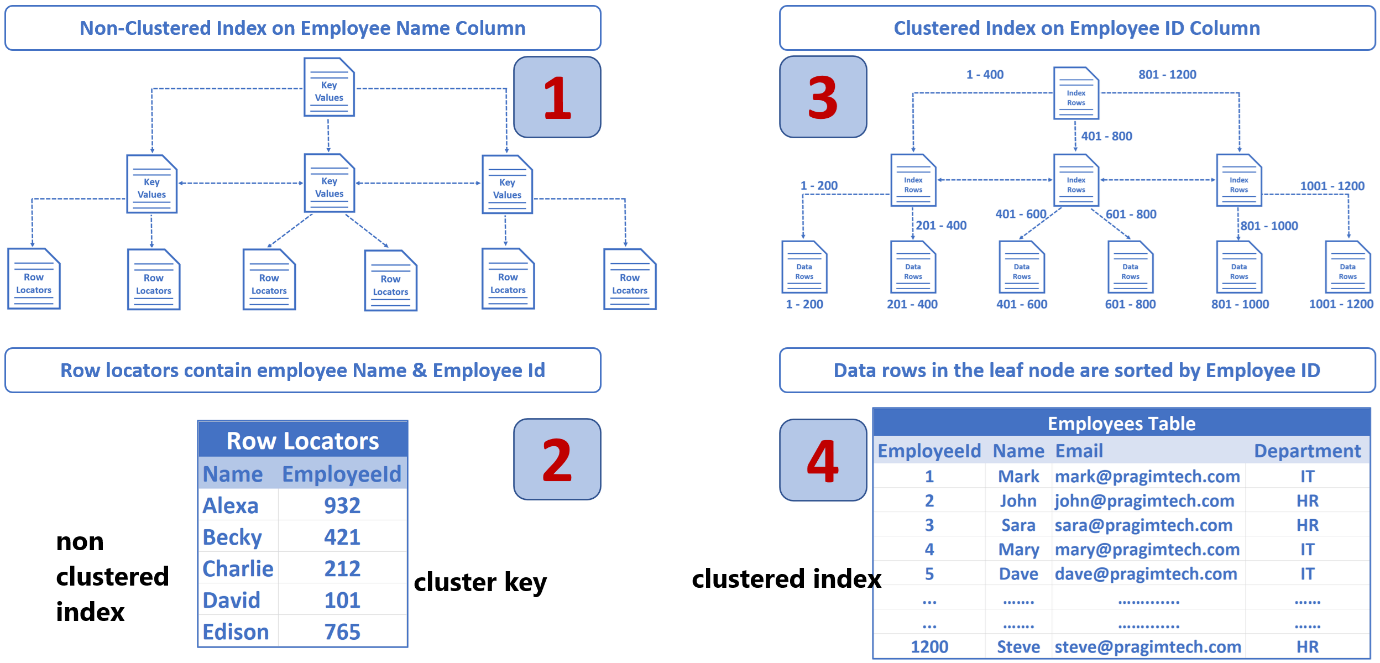
This defines a **one-to-many** relationship between Customers and Orders — each customer can have multiple orders.

### Indexes: Data Structures for Efficient Data Retrieval in DBMS

Indexes in a DBMS are special data structures that improve the speed of data retrieval operations on a database table, at the cost of additional storage and maintenance. They function similarly to a book's index, allowing the database to locate data quickly without scanning every row in the table.

* **Clustered Index**:
  + The table's rows are stored in the order of the index keys.
  + Each table can have only **one** clustered index because the data is physically sorted based on that index.
  + The clustered index essentially determines how the rows are stored on disk.
  + Example: Primary key indexes are often clustered, ensuring fast data retrieval by primary key.
* **Non-Clustered Index**:
  + The data is stored separately from the index, and the index contains pointers to the data rows.
  + A table can have multiple non-clustered indexes.
  + Non-clustered indexes improve retrieval times for queries that use columns other than those in the primary key.

Indexes improve query performance, especially for large datasets, but they require additional overhead for storage and updating the index during data modifications (inserts, updates, or deletes).



**⭐** **⭐** **⭐** **https://www.pragimtech.com/blog/sql-optimization/how-do-sql-indexes-work/**

**What is normalization. Give the meaning of 1NF, 2NF, 3NF**

Normalization – read the ipynb file

Database normalization is the process of organizing data in a relational database to **reduce redundancy and improve data integrity**. The normalization process involves dividing large tables into smaller, more manageable tables and defining relationships between them.

**First Normal Form (1NF):**

Objective: Eliminate duplicate data and ensure each column holds atomic values.

Requirements: Each table cell must contain a single, indivisible value. All entries in a column must be of the same data type.

**Second Normal Form (2NF):**

Objective: Eliminate partial dependencies on composite primary keys. “A\_B” key

Requirements: Must be in 1NF, and no non-prime attribute (attribute not part of the primary key) should be functionally dependent on only a part of the primary key.

**Third Normal Form (3NF):**

Objective: Eliminate transitive dependencies.

Requirements: Must be in 2NF, and no non-prime attribute should be transitively dependent on the primary key. In other words, no non-prime attribute should depend on another non-prime attribute.

**Concurrency Control in DBMS:**

Concurrency control refers to the mechanisms and techniques implemented to manage and coordinate access to the database by multiple transactions concurrently. The goal is to ensure that transactions can execute simultaneously without causing data inconsistencies or conflicts. Concurrency control plays a crucial role in maintaining the consistency and integrity of the database in a multi-user environment.

**Types of Concurrency Control:**

**Lock-Based Protocols**:

* **Shared Lock (S)**: Allows multiple transactions to read a resource.
* **Exclusive Lock (X)**: Allows only one transaction to write or modify a resource.

**Timestamp-Based Protocols**:

* Assigns each transaction a unique timestamp.
* Ensures transactions execute in timestamp order.
* Prevents conflicts by comparing timestamps for read/write operations.

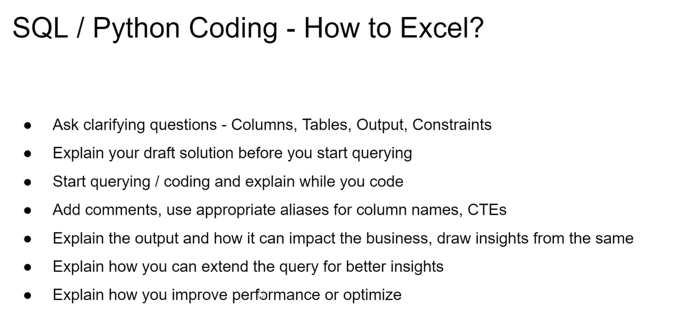
**Multiversion Concurrency Control (MVCC)**:

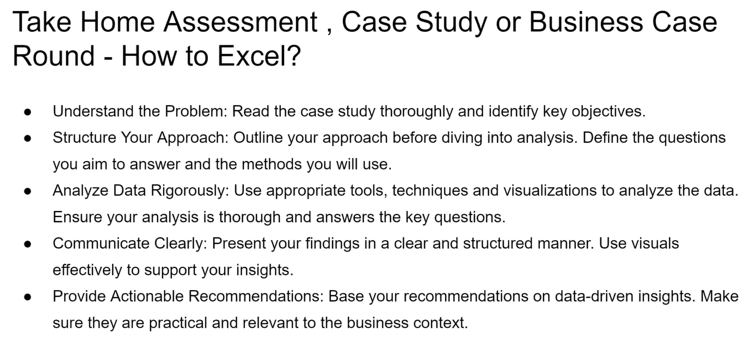
* Maintains multiple versions of data.

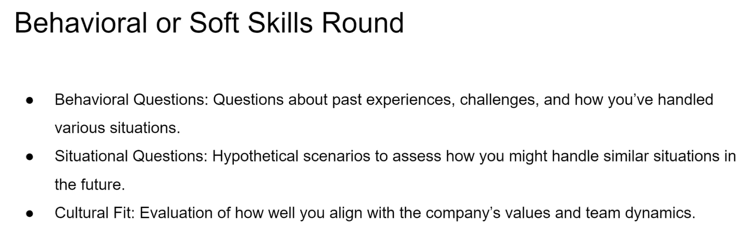
**Validation-Based Protocols**:

* Divides transactions into three phases:
  + **Read Phase**: Transaction reads data and performs computations.
  + **Validation Phase**: Checks for conflicts with other transactions.
  + **Write Phase**: Updates the database if validation passes.

Case Studies







How do you ensure the accuracy and reliability of your SQL queries?

1. **Testing in Stages**: Break down complex queries test each part separately before combining them.
2. **Use Sample Data:** Test queries initially on a subset of the **Check Execution Plans:** Examine the execution plans to understand how the database engine is processing the query. **Optimize queries** for better performance.
3. Avoid SELECT \*:
4. **Use Parameters**: Use parameterized queries to prevent SQL injection and improve query plan caching.
5. **Data Integrity Checks**: Implement checks for data integrity, ensuring that the results match expectations based on business rules.
6. **Consistent Formatting:** Follow a consistent formatting style to improve code readability and maintenance.

Did you perform regular recovery and backups

#### **Backup Steps:**

1. **Identify Critical Data:** Determine the files, databases, or systems that are essential for business continuity.
2. **Choose Backup Method:** Use full, incremental, or differential backup strategies depending on the frequency of data changes. Utilize tools like cloud services, on-premise storage, or hybrid solutions.
3. **Schedule Backups:** Automate backups using software or scripts to ensure consistency (e.g., daily, weekly). Use tools like **cron jobs** (Linux) or **Task Scheduler** (Windows).
4. **Verify Backup Integrity:** Periodically test backups to confirm they are complete and functional.
5. **Maintain Backup Redundancy:** Store backups in multiple locations (e.g., local, cloud, offsite) to mitigate risks of physical damage.
6. **Document Backup Procedures:** Maintain clear guidelines on backup processes for team members.

#### **Recovery Steps:**

1. **Assess the Incident:** Identify the scope and cause of data loss or corruption.
2. **Retrieve Latest Backup:** Locate the most recent functional backup that matches recovery needs.
3. **Restore Data:** Use backup tools or platforms to restore data to the original or alternative system. For databases, use commands like RESTORE DATABASE (SQL Server) or mysqlimport (MySQL).
4. **Test Restored Data:** Verify the integrity and functionality of restored data to ensure no corruption.
5. **Implement Preventive Measures:** Analyze the cause of data loss and update backup policies or infrastructure to avoid recurrence.
6. **Update Documentation:** Record the incident and recovery process for reference and improvement.

Can you describe a project where you used SQL for data analysis? What types of queries did you write?

SQL is always in combination with visualization tools.

Nested SQL statements: 2 joins, 5 parameters for Y Q M Role etc

Normalization, query reduction, indexing,  primary / foreign key

Performance Issues

**Issue: Slow Query Execution Times**

* **Challenge:** Complex queries with multiple joins and large datasets were slow to execute.
* **Solution:**
  + Used **indexes** on frequently queried columns to speed up data retrieval.
  + Optimized queries by reducing unnecessary joins and using EXISTS instead of IN where appropriate.
  + Analyzed query plans with tools like EXPLAIN to identify bottlenecks and restructure the queries.

 **Issue: Deadlocks in Concurrent Transactions**

* **Challenge:** Multiple users accessing and updating the same records caused deadlocks.
* **Solution:**
  + Implemented **locking strategies**, such as row-level locking instead of table-level locking, to minimize contention.
  + Ordered transactions consistently to avoid circular wait conditions.
  + Used retry logic to handle deadlocks gracefully.

 **Issue: High Disk I/O and Slow Disk Performance**

* **Challenge:** Queries involving large table scans overloaded the disk I/O.
* **Solution:**
  + Partitioned large tables to improve query performance.
  + Moved frequently accessed data to in-memory caches like Redis or Memcached.
  + Archived old or rarely accessed data to reduce the active dataset size.

 **Issue: Inefficient Use of Temporary Tables**

* **Challenge:** Overuse of temporary tables caused resource contention and slowed performance.
* **Solution:**
  + Reduced temporary table usage by leveraging **Common Table Expressions (CTEs)** or inline views.
  + Ensured temporary tables were indexed where necessary.

 **Issue: Suboptimal Database Schema Design**

* **Challenge:** Poorly normalized tables resulted in redundant data and complicated queries.
* **Solution:**
  + Refactored the schema to follow normalization principles, eliminating redundancy.
  + Used denormalized structures for read-heavy use cases and added precomputed columns or materialized views for faster querying.