Database Systems Cheatsheet

A comprehensive cheat sheet covering essential concepts in database systems, including data modeling, SQL, normalization, transactions, and indexing.

Data Modeling

Entity-Relationship (ER) Model

Entity: A real-world object distinguishable from other objects.

Example: Customer, Product, Order

Attribute: A property describing an entity.

Example: Customer ID, Product Name, Order Date

Relationship: An association among entities.

Example: Customer places Order, Product is part of Order

Cardinality: Specifies the number of instances of one entity that can be related to another entity.

Types: One-to-one (1:1), One-to-many (1:N), Many-to-one (N:1), Many-to-many (N:M)

Primary Key: A unique identifier for an entity.

Example: Customer ID in Customer entity

Foreign Key: An attribute in one entity that refers to the primary key of another entity, establishing a link between them.

Example: Customer ID in Order entity referencing Customer entity

Enhanced Entity-Relationship (EER) Model

Specialization:	Creating subtypes (child entities) from a supertype (parent entity). **Example:** Employee (supertype) can be specialized into Salaried_Employee and Hourly_Employee (subtypes).
Generalization:	Creating a supertype from subtypes. **Example:** Combining Car and Truck into Vehicle (supertype).
Aggregation:	Treating a relationship as an entity. **Example:** Project entity consisting of Worker entity and Task entity.
Inheritance:	Subtypes inherit attributes and relationships from their supertype. **Example:** Salaried_Employee inherits attributes like Employee ID and Name from Employee.

UML Class Diagrams

Class: Represents a set of objects with common attributes and behavior.

Example: `Customer` class with attributes `CustomerID`, `Name`, `Address`.

Association: Represents a relationship between classes.

Example: `Customer` *places* `Order`.

Multiplicity: Specifies the cardinality of the association.

Example: One `Customer` can place many `Order`s (1..*).

Aggregation/Composition: Represents a part-whole relationship.

Example: `Order` *consists of* `OrderItem`s (composition if `OrderItem` cannot exist without `Order`).

SQL Fundamentals

Basic Queries

`SELECT` statement:	Retrieves data from a database. **Example:** `SELECT column1, column2 FROM table_name;`
`WHERE` clause:	Filters the results based on a condition. **Example:** `SELECT * FROM Customers WHERE Country = 'USA';`
`ORDER BY` clause:	Sorts the results. **Example:** `SELECT * FROM Products ORDER BY Price DESC;`
`LIMIT` clause:	Limits the number of rows returned. **Example:** `SELECT * FROM Employees LIMIT 10;`
`DISTINCT` keyword:	Retrieves unique values. **Example:** `SELECT DISTINCT Country FROM Customers;`

Joins

`INNER JOIN`:

Returns rows when there is a match in both tables.

Example:

```
SELECT Orders.OrderID,
Customers.CustomerName
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID;
```

`LEFT JOIN` (or `LEFT OUTER JOIN`):

Returns all rows from the left table, and the matched rows from the right table. If there is no match, the result is NULL on the right side.

Example:

```
SELECT Customers.CustomerName,
Orders.OrderID
FROM Customers
LEFT JOIN Orders ON
Customers.CustomerID =
Orders.CustomerID;
```

`RIGHT JOIN` (or `RIGHT OUTER JOIN`):

Returns all rows from the right table, and the matched rows from the left table. If there is no match, the result is NULL on the left side.

Example:

```
SELECT Customers.CustomerName,
Orders.OrderID
FROM Customers
RIGHT JOIN Orders ON
Customers.CustomerID =
Orders.CustomerID;
```

`FULL OUTER JOIN`:

Returns all rows when there is a match in one of the tables.

Example:

```
SELECT Customers.CustomerName,
Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON
Customers.CustomerID =
Orders.CustomerID;
```

Aggregate Functions

`COUNT()` - Returns the number of rows.

Example:

```
SELECT COUNT(*) FROM Orders;
```

`SUM()` - Returns the sum of values.

Example:

```
SELECT SUM(Price) FROM Products;
```

`AVG()` - Returns the average value.

Example:

```
SELECT AVG(Price) FROM Products;
```

`MIN()` - Returns the minimum value.

Example:

```
SELECT MIN(Price) FROM Products;
```

`MAX()` - Returns the maximum value.

Example:

```
SELECT MAX(Price) FROM Products;
```

Normalization

Normal Forms

1NF (First Normal Form):
Eliminate repeating groups of data.
Each column should contain only atomic values.

2NF (Second Normal Form):
Must be in 1NF and eliminate redundant data.
No non-key attribute should be dependent on a proper subset of any candidate key.

3NF (Third Normal Form):
Must be in 2NF and eliminate transitive dependencies.
No non-key attribute should be transitively dependent on the primary key.

BCNF (Boyce-Codd Normal Form):
A stronger version of 3NF.
Every determinant must be a candidate key.

4NF (Fourth Normal Form):
Must be in BCNF and eliminate multi-valued dependencies.

5NF (Fifth Normal Form):
Must be in 4NF and eliminate join dependencies.

Example of Normalization

Consider a table `Orders` with columns: `OrderID`, `CustomerID`, `CustomerName`, `CustomerAddress`, `ProductID`, `ProductName`, `ProductPrice`.

Unnormalized:

```
OrderID | CustomerID | CustomerName | CustomerAddress | ProductID |
ProductName | ProductPrice
--------|------------|--------------|-----------------|-----------|-------
------|------------
1       | 101        | John Doe     | 123 Main St     | 1         | Laptop
| 1200
1       | 101        | John Doe     | 123 Main St     | 2         | Mouse
| 25
```

1NF:
Remove repeating groups by creating separate rows for each product.

```
OrderID | CustomerID | CustomerName | CustomerAddress | ProductID |
ProductName | ProductPrice
--------|------------|--------------|-----------------|-----------|-------
------|------------
1       | 101        | John Doe     | 123 Main St     | 1         | Laptop
| 1200
1       | 101        | John Doe     | 123 Main St     | 2         | Mouse
| 25
```

2NF:
Create separate tables for `Customers`, `Products`, and `Orders` to eliminate redundant data.

Tables:
`Customers`: `CustomerID`, `CustomerName`, `CustomerAddress`
`Products`: `ProductID`, `ProductName`, `ProductPrice`
`Orders`: `OrderID`, `CustomerID`, `ProductID`

Transactions and Indexing

Transaction Properties (ACID)

Atomicity: All operations in a transaction must be treated as a single “unit”. Either all operations succeed, or none do.

Example: Transferring money from one account to another involves debiting one account and crediting another. Both must succeed or fail together.

Consistency: A transaction must maintain the integrity of the database. Moving from one valid state to another.

Example: A transaction should not violate any defined constraints (e.g., primary key, foreign key).

Isolation: Transactions should be isolated from each other. Concurrent execution should have the same result as if transactions were executed serially.

Example: Two transactions updating the same data should not interfere with each other.

Durability: Once a transaction is committed, the changes are permanent and will survive system failures.

Example: After a successful money transfer, the changes should not be lost even if the system crashes immediately afterward.

Transaction Management

`START TRANSACTION`:	Begins a new transaction. **Example:** `START TRANSACTION;`
`COMMIT`:	Saves the changes made during the transaction. **Example:** `COMMIT;`
`ROLLBACK`:	Undoes the changes made during the transaction. **Example:** `ROLLBACK;`
`SAVEPOINT`:	Creates a point within a transaction to which you can rollback. **Example:** `SAVEPOINT my_savepoint;`
`RELEASE SAVEPOINT`:	Removes a previously defined savepoint. **Example:** `RELEASE SAVEPOINT my_savepoint;`

Indexing

Purpose:
Indexes improve the speed of data retrieval operations on a database table.

Types:

- **B-tree index:** Most common type, efficient for range queries and equality lookups.
- **Hash index:** Fast for equality lookups but not suitable for range queries.
- **Full-text index:** Used for searching text data.

Creating an Index:

```
CREATE INDEX index_name ON table_name
(column1, column2, ...);
```

Example:

```
CREATE INDEX idx_customer_name ON Customers
(CustomerName);
```

Considerations:
Indexes can slow down write operations (INSERT, UPDATE, DELETE) because the index also needs to be updated. Choose indexes wisely based on the most frequent queries.