## **Database Management Deep Dive**

Here's a breakdown of the topics you mentioned:

**1. Database:**

* An organized collection of structured data electronically stored in a computer system.
* Managed by a Database Management System (DBMS) software for storing, retrieving, and manipulating data efficiently.
* Common organization involves tables with rows and columns for data and relationships.
* Used in various applications like library systems, online stores, and financial transactions.

**2. SQL (Structured Query Language):**

* A specific programming language for interacting with relational databases.
* Used to create, manipulate, and retrieve data from databases.
* Offers commands for various functionalities like:
  + **Data Definition Language (DDL):** Creating and altering database structures (tables, indexes, etc.)
  + **Data Manipulation Language (DML):** Inserting, updating, and deleting data.
  + **Data Query Language (DQL):** Selecting and retrieving specific data.

**3. SQL Server:**

* A popular relational database management system (DBMS) developed by Microsoft.
* Uses Transact-SQL (T-SQL) as its primary query language, an extension of standard SQL.
* Widely used for business applications, data warehousing, and analytics.

**4. Basic T-SQL Commands (Focusing on the first 4):**

* T-SQL (Transact-SQL) is a Microsoft extension of standard SQL for working with SQL Server. Here are some fundamental commands (assuming you start with 1):
  + **SELECT:** Retrieves data from one or more tables.
  + **INSERT:** Adds new data into a table.
  + **UPDATE:** Modifies existing data in a table.
  + **DELETE:** Removes data from a table.

**5. SQL Server Management Studio (SSMS):**

* A graphical tool from Microsoft for managing and administering SQL Server databases.
* Allows users to:
  + Write and execute T-SQL queries.
  + Design and modify database structures.
  + Manage users, security, and backups.

**6. Database Schema:**

* A blueprint that defines the structure of a database, including tables, columns, data types, and relationships between them.
* Provides organization and ensures data consistency.

**7. Data Types and Null, Not Null options, Identity Columns:**

* **Data Types:** Specify the type of data a column can hold (e.g., text, numbers, dates).
* **NULL:** Represents missing or unknown data in a column.
* **NOT NULL:** Enforces that a column cannot have missing values.
* **Identity Columns:** Automatically generate unique sequential numbers for each new row inserted.

**8. Working with Data:**

* This encompasses various tasks like:
  + **Importing and Exporting Data:** Moving data between databases or other systems.
  + **Filtering and Sorting Data:** Extracting specific data subsets based on criteria.
  + **Joining Tables:** Combining data from multiple tables based on relationships.  
    pen\_spark

**SQL Server DDL Commands for Creating and Altering Databases and Tables**

Here's a breakdown of essential SQL Server DDL commands for managing databases and tables:

**1. CREATE DATABASE:** This command creates a new database:

SQL

CREATE DATABASE [DatabaseName];

Use code with caution.

content\_copy

**Example:**

SQL

CREATE DATABASE MyNewDatabase;

Use code with caution.

content\_copy

**2. CREATE TABLE:** This command creates a new table within a database:

SQL

CREATE TABLE [TableName] (

[ColumnName1] [DataType]([Size]), -- Define column with data type and size (optional)

[ColumnName2] [DataType]([Size]), -- Add more columns with data types and sizes

...

CONSTRAINT [ConstraintName] PRIMARY KEY ([ColumnList]) -- Optional primary key constraint

);

Use code with caution.

content\_copy

**Example:**

SQL

CREATE TABLE Customers (

CustomerID int PRIMARY KEY IDENTITY, -- Auto-incrementing integer primary key

CustomerName varchar(50) NOT NULL,

Email varchar(100),

Phone varchar(20)

);

Use code with caution.

content\_copy

**3. ALTER TABLE:** This command modifies the structure of an existing table. Here are some common variations:

\* \*\*Adding a new column:\*\*

```sql

ALTER TABLE [TableName] ADD [NewColumnName] [DataType]([Size]);

```

\* \*\*Modifying an existing column:\*\* (e.g., changing data type)

```sql

ALTER TABLE [TableName] ALTER COLUMN [ColumnName] [NewDataType]([Size]);

```

\* \*\*Dropping a column:\*\*

```sql

ALTER TABLE [TableName] DROP COLUMN [ColumnName];

```

**4. DROP TABLE:** This command removes a table entirely:

SQL

DROP TABLE [TableName];

Use code with caution.

content\_copy

**Important Notes:**

* Remember to replace [DatabaseName] and [TableName] with your desired names.
* Square brackets ([]) are used to enclose optional elements.
* Data types (varchar, int, etc.) and sizes specify the format and storage requirements for data in each column.
* Primary key constraints ensure unique identification for each row in the table.
* Use ALTER TABLE cautiously, as modifications can impact existing data.

**Additional DDL Commands:**

* **CREATE INDEX:** Creates an index to improve query performance.
* **CREATE VIEW:** Creates a virtual table based on a query.
* **DROP DATABASE:** Removes a database (be cautious, as data will be lost).

**Data types in Sql**  
  
SQL Server offers a rich set of data types to cater to various data storage needs. Here's a breakdown of some common data types along with their practical uses and examples:

**Exact Numerics:**

* **int:** This is a versatile integer data type that can hold whole numbers ranging from -2,147,483,648 to 2,147,483,647. It's ideal for storing quantities, IDs, and other whole number values.
  + **Example:** CustomerID int in a customer table.
* **decimal/numeric:** These data types provide high precision for storing decimal numbers. They are specified as decimal(p, s), where 'p' represents the total number of digits and 's' represents the number of digits to the right of the decimal point. Use them for financial calculations, measurements, or any scenario requiring precise decimal values.
  + **Example:** Price decimal(10, 2) in a product table to store prices up to two decimal places.
* **bit:** This data type represents a single binary bit, either 0 or 1. It's suitable for storing Boolean flags (true/false) or binary values.
  + **Example:** IsActive bit in a user table to indicate active (1) or inactive (0) user status.

**Approximate Numerics:**

* **float/real:** These data types store single-precision (float) and double-precision (real) floating-point numbers for representing real numbers with a wider range but less precision compared to exact numerics. They are useful for scientific calculations or engineering data.
  + **Example:** Temperature float in a weather data table to store temperature readings.

**Date and Time:**

* **date:** This data type stores only the date portion (year, month, day). Use it for birth dates, anniversaries, or any scenario where only the date information is relevant.
  + **Example:** HireDate date in an employee table.
* **datetime/smalldatetime:** These data types store both date and time information with varying precisions. datetime offers the highest precision, while smalldatetime is more space-efficient for dates and times without milliseconds.
  + **Example:** OrderPlaced datetime in an order table to capture the exact time an order was placed.

**Character Strings:**

* **varchar(n):** This is a variable-length character string data type that can store text data up to a specified maximum length 'n'. It's efficient for storing text data of varying lengths, like names, addresses, or descriptions.
  + **Example:** CustomerName varchar(50) in a customer table.
* **nvarchar(n):** Similar to varchar, but for storing Unicode characters, which allows for a wider range of characters beyond the basic Latin alphabet.
  + **Example:** ProductDescription nvarchar(max) for storing product descriptions that might include characters from different languages.

**Other Data Types:**

* **money/smallmoney:** These data types are specifically designed for storing currency values, taking into account regional monetary conventions.
  + **Example:** UnitPrice money in a product table.
* **varbinary(n):** This data type stores variable-length binary data, useful for images, audio files, or any non-textual data.
  + **Example:** ProductImage varbinary(max) to store product image data.

This is just a glimpse into the world of SQL Server data types. Remember to choose the data type that best suits the kind of data you're storing to optimize storage space and ensure data integrity.

Delete vs Truncate  
  
**Transaction Log:**

* **TRUNCATE:** This operation is a Data Definition Language (DDL) statement. It deallocates the data pages used by the table and only logs this deallocation in the transaction log. This makes it faster than DELETE.
* **DELETE:** This operation is a Data Manipulation Language (DML) statement. It works by row-by-row deletion and records each deleted row in the transaction log. This logging process can be slower, especially for larger tables.

Data pages are the fundamental unit of data storage for tables in SQL Server. They act like the building blocks that hold the actual data stored in your tables. Here's a detailed explanation:

* **Data Organization:** When you create a table, SQL Server doesn't store the data as a single, continuous stream. Instead, it breaks it down into smaller, manageable units called data pages.
* **Page Size:** Each data page has a fixed size, typically 8 KB (8,192 bytes) by default in SQL Server. This size ensures efficient storage and retrieval of data.
* **Page Structure:** A data page is further divided into three sections:
  + **Page Header:** Stores information about the page itself, like its identification number, type (data page, index page, etc.), and free space availability.
  + **Payload:** This is the most crucial part, where the actual data rows from your table are stored. The format and size of data stored here depend on the data types used in your table columns.
  + **Row Offset Array:** This section keeps track of the starting positions of each data row within the payload, allowing for quick retrieval of specific rows.

**Benefits of Data Pages:**

* **Efficient Storage:** Data pages allow for efficient allocation and management of disk space since data is stored in fixed-size chunks.
* **Parallel Processing:** SQL Server can access and process multiple data pages simultaneously, improving query performance.
* **Granularity:** Data pages enable granular control over data access. SQL Server only needs to read or write specific data pages instead of the entire table, reducing I/O operations.

**Architecture of Data Pages**  
  
**Three Sections:** A data page is divided into three main sections:

* **Page Header (96 bytes):** This section stores critical metadata about the page itself. It includes information like:
  + Page type (data page, index page, etc.)
  + Location within the data file
  + Amount of free space available on the page
  + Pointer to the next or previous data page (relevant for extents)
* **Payload (variable size):** This is the core section where the actual data for your table rows is stored. The amount of data that can fit here depends on the size and structure of each row.
* **Row Offset Array (variable size):** This section acts like an index for the payload. It keeps track of the starting position of each row within the payload, allowing SQL Server to quickly locate specific rows on the page.

**Non-clustered index**  
  
Suppose you have a Customers table with columns CustomerID (primary key), CustomerName, Email, and City. You want to create a non-clustered index on the CustomerName column in ascending order to speed up searches by customer name:

SQL

CREATE NONCLUSTERED INDEX IX\_CustomerName\_ASC

ON Customers (CustomerName ASC);

**Explanation:**

* This code creates a non-clustered index named IX\_CustomerName\_ASC on the Customers table.
* The index includes only the CustomerName column in ascending order.

**Clustered-index**  
  
By default, in SQL Server, the primary key of a table acts as a clustered index **if** a clustered index hasn't already been explicitly created on the table.

Sql Commands  
**SQL Commands with Examples**

Here are some common SQL commands with examples to get you started:

**1. SELECT:** This command retrieves data from a table.

SQL

SELECT \* FROM customers; -- Select all columns (\*) from the "customers" table.

SELECT customer\_name, email FROM customers WHERE city = 'New York';

-- Select specific columns (customer\_name, email) from "customers" where the "city" is 'New York'.

Use code with caution.

content\_copy

**2. INSERT:** This command adds new data to a table.

SQL

INSERT INTO customers (customer\_name, email, phone) VALUES ('John Doe', 'john.doe@example.com', '123-456-7890');

-- Insert a new row into "customers" table with specific values.

INSERT INTO Customers (CustomerID, CustomerName, City)

VALUES (100, 'John Doe', 'Seattle');

**3. UPDATE:** This command modifies existing data in a table.

SQL

UPDATE customers SET email = 'jane.doe@example.com' WHERE customer\_id = 1;

-- Update the email address for a customer with ID 1 in the "customers" table.

Use code with caution.

content\_copy

**4. DELETE:** This command removes data from a table.

SQL

DELETE FROM customers WHERE last\_name = 'Smith';

-- Delete all rows from "customers" where the "last\_name" is 'Smith'.

Use code with caution.

content\_copy

**5. WHERE Clause:** This clause specifies a condition to filter the data retrieved by the SELECT statement.

**6. ORDER BY:** This clause sorts the result set of a SELECT statement in ascending or descending order based on a column.

SQL

SELECT \* FROM products ORDER BY price DESC; -- Select all products from "products" ordered by price (highest to lowest).

Types of Sql Server   
  
**Types of SQL Server (Editions):**

While SQL Server itself functions as a single server application, it comes in various editions offering different feature sets and functionalities. Here's a breakdown of the common editions:

1. **SQL Server Enterprise:**
   * The most feature-rich edition designed for demanding workloads.
   * Ideal for mission-critical applications, high availability setups, and large-scale data warehousing.
   * Supports advanced capabilities like In-Memory Online Transaction Processing (OLTP) and Machine Learning Services.
2. **SQL Server Standard:**
   * A well-balanced edition suitable for mid-tier applications and data marts.
   * Offers core database management features like replication, backup, and basic reporting services.
   * A good choice for businesses with moderate data volumes and application needs.
3. **SQL Server Developer:** (Free for development and testing)
   * Provides all the features of the Enterprise edition but with limitations on licensing for production use.
   * Ideal for developers to build and test applications before deployment.
4. **SQL Server Express:** (Free, entry-level)
   * Suitable for learning and small-scale applications.
   * Has limitations on database size, memory usage, and processing power.
   * A good starting point for learning SQL Server and developing desktop or small server-based applications.

**Choosing the right edition** depends on your specific needs such as:

* Database size and complexity
* Processing power requirements
* Features required (reporting, analytics, etc.)
* Deployment environment (on-premises or cloud)

There are two main free options available for SQL Server:

1. **SQL Server Express:** This is the most popular free edition. It's ideal for development, testing, and small-scale production environments. Here are some key points about SQL Server Express:
   * **Free to use:** No licensing cost involved.
   * **Limitations:**
     + Database size limit: Up to 10 GB per database file (with a total limit of 10 GB)
     + Memory limit: Up to 1 GB of RAM
     + Limited CPU usage: Restricted to a single CPU core
     + Lacks some advanced features found in paid editions (e.g., replication, high availability)
   * **Use cases:** Learning SQL Server, developing and testing applications, small business applications with low data volume.
2. **SQL Server Developer:** This edition offers all the features of the Enterprise edition, making it a powerful tool for developers. However, it has limitations on how it can be used:
   * **Free to use:** But only for development and testing purposes.
   * **Limitations:**
     + Not licensed for production use. Data created in this edition cannot be directly transferred to a production environment without acquiring a paid license.
     + Limited to a single user with local access.
   * **Use cases:** Building and testing complex database applications before deploying them to a production environment with a paid edition.

**Here's a quick comparison to help you decide:**

| **Feature** | **SQL Server Express** | **SQL Server Developer** |
| --- | --- | --- |
| Cost | Free | Free |
| Use Case | Development, testing, small production | Development, testing |
| Limitations | Database size, memory, CPU, features | Not for production, single user |

**Pricing Details:** (Note: Prices can vary depending on specific versions and vendors, so consider this a general overview)

* **SQL Server Enterprise:** As the most feature-rich edition, Enterprise typically has the highest cost. It's priced based on the number of cores on the server using the core-based licensing model.
* **SQL Server Standard:** This edition is priced lower than Enterprise, also based on the number of cores on the server. Since it offers fewer features, it's generally more affordable.

**Here's a table summarizing the key differences in pricing:**

| **Feature** | **SQL Server Enterprise** | **SQL Server Standard** |
| --- | --- | --- |
| Licensing Model | Core-based or Server/CAL | Core-based or Server/CAL |
| Price | Higher (due to more features) | Lower |
| Factors Affecting Price | Number of cores | Number of cores  pen\_spark |

**Master:**

* **The Brain of the Server:** The master database acts as the central repository for all system-wide information on the SQL Server instance. It stores details like login accounts, server configuration settings, and metadata for all other databases on the server.
* **Critical for Startup:** SQL Server cannot start without a functional master database. Since it holds information on how the server is configured, its absence would leave the server inoperable.

**Model:**

* **The Blueprint for New Databases:** The model database serves as a template for creating new user databases on the SQL Server instance. It contains default settings and objects, like security settings and system objects, which are copied to new databases when they're created.
* **Customizing the Template:** You can modify certain aspects of the model database, such as database size, recovery model, and collation. These changes will then be applied to any new databases created afterward.

**MSDB:**

* **The Automation Hub:** The msdb database is used by SQL Server Agent, a built-in automation tool. It stores information related to scheduled tasks, backups, maintenance plans, and alerts.
* **Keeping Track of Operations:** If you create a database maintenance plan to take daily backups, for instance, the details of that plan are stored in the msdb database. This allows you to monitor and manage your automated tasks efficiently.

**TempDB:**

* **The Temporary Workspace:** The tempdb database provides temporary storage space for objects used by SQL Server during various operations. These objects include temporary tables, sort results, and intermediate calculations.
* **A Fresh Start Every Time:** Unlike other databases, tempdb has a unique characteristic. Its contents are automatically cleared whenever the SQL Server service is restarted. This ensures a clean slate for temporary objects during each new session.

Link - https://www.w3schools.com/sql/sql\_insert.asp