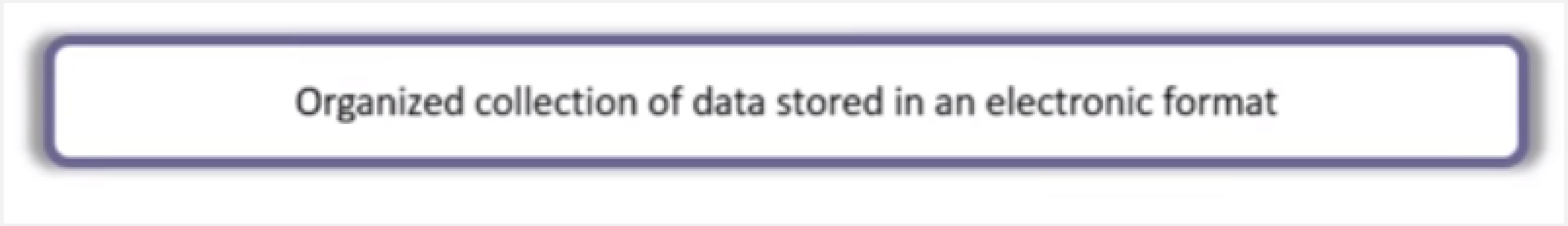
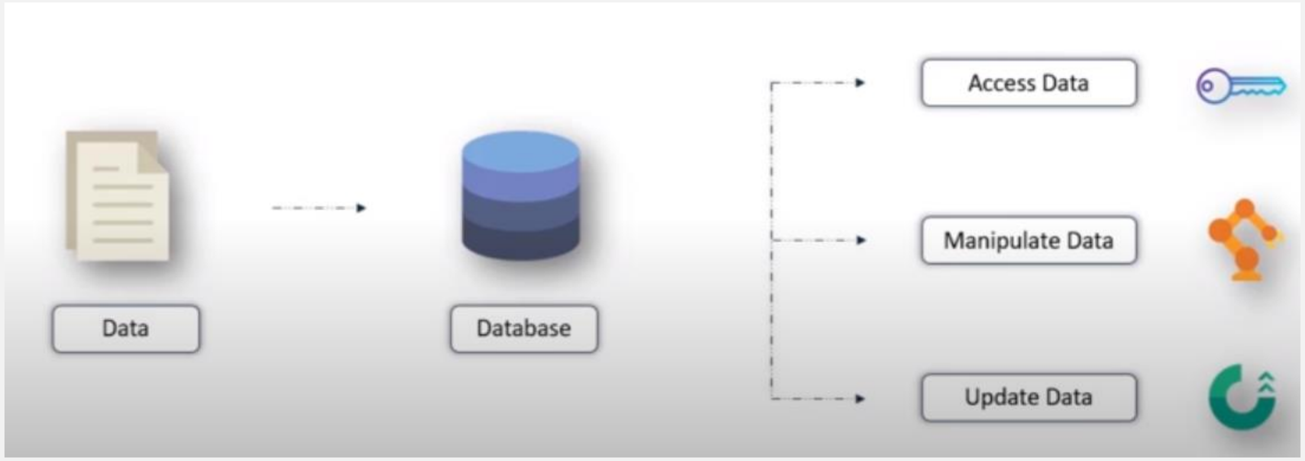
MYSQL Basics

Archer Infotech , PUNE

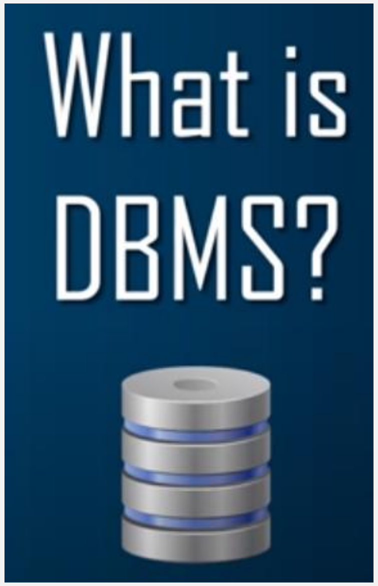
What is Database ?


Organized collection of data stored in an electronic format



A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management easy

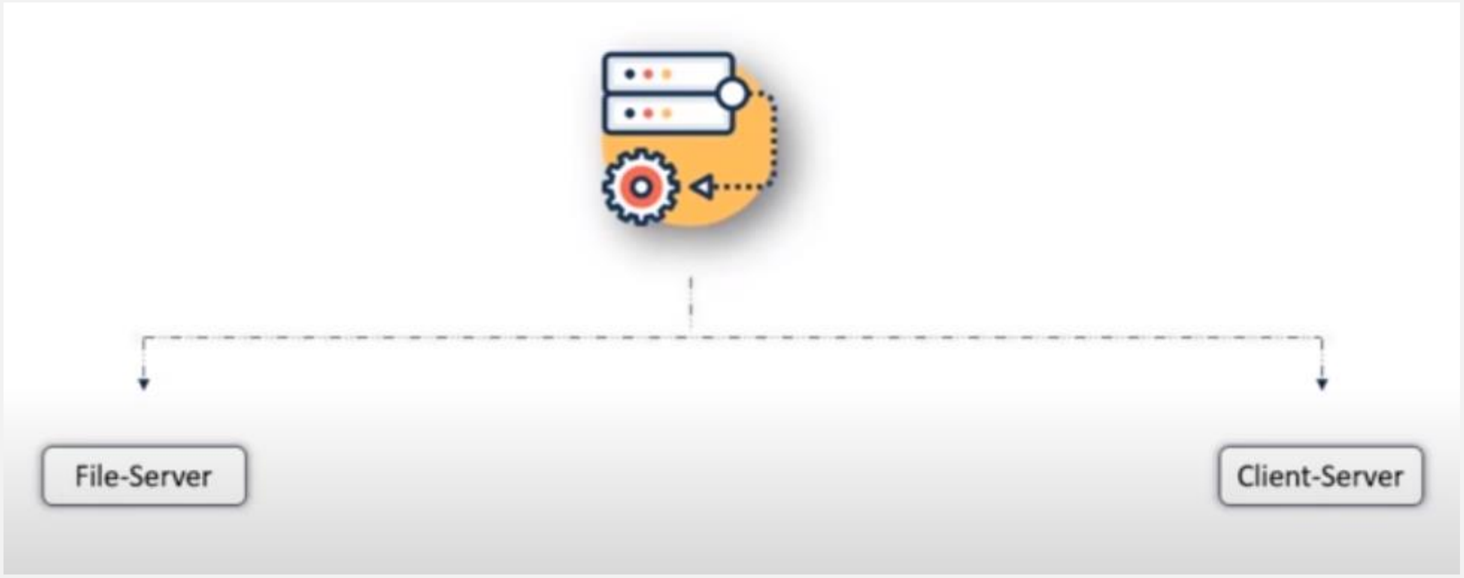
What is DBMS ?



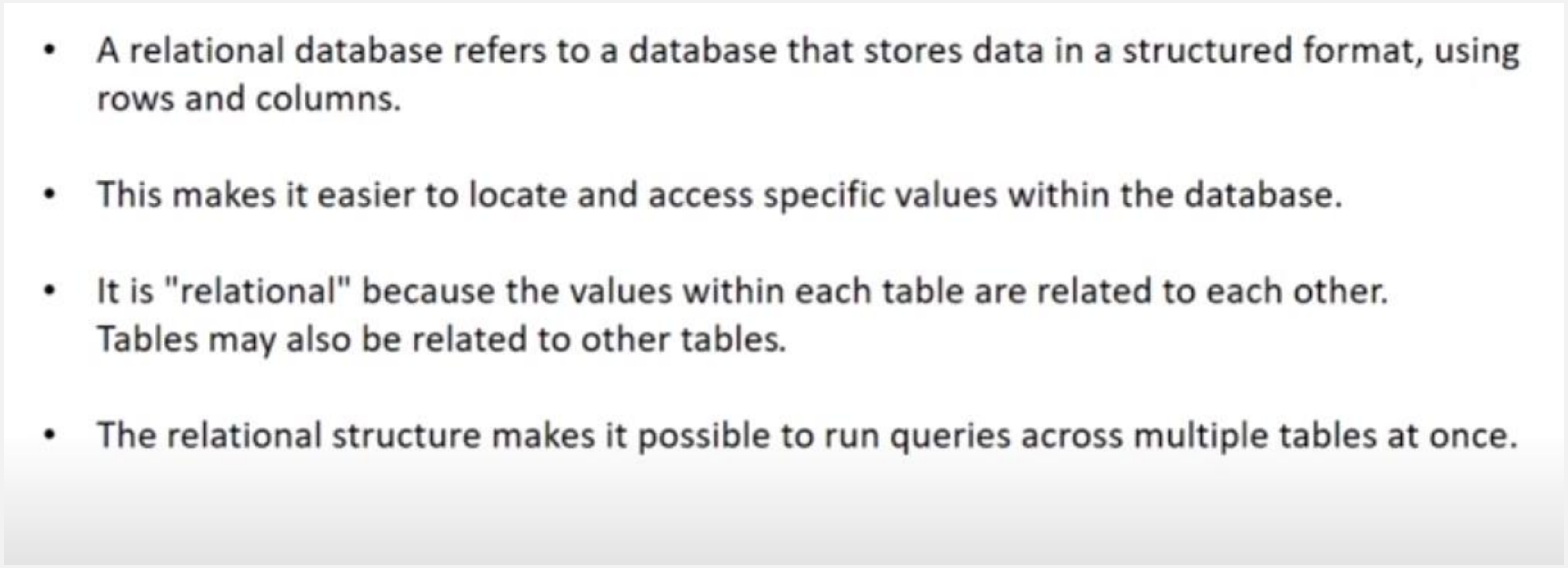
A Database Management System (DBMS) is a software that is used to manage the Database.

A DBMS basically serves as an interface between the database and its end-users or programs, allowing users to retrieve, update, and manage how the information is organized and optimized.

Types of Database Architecture

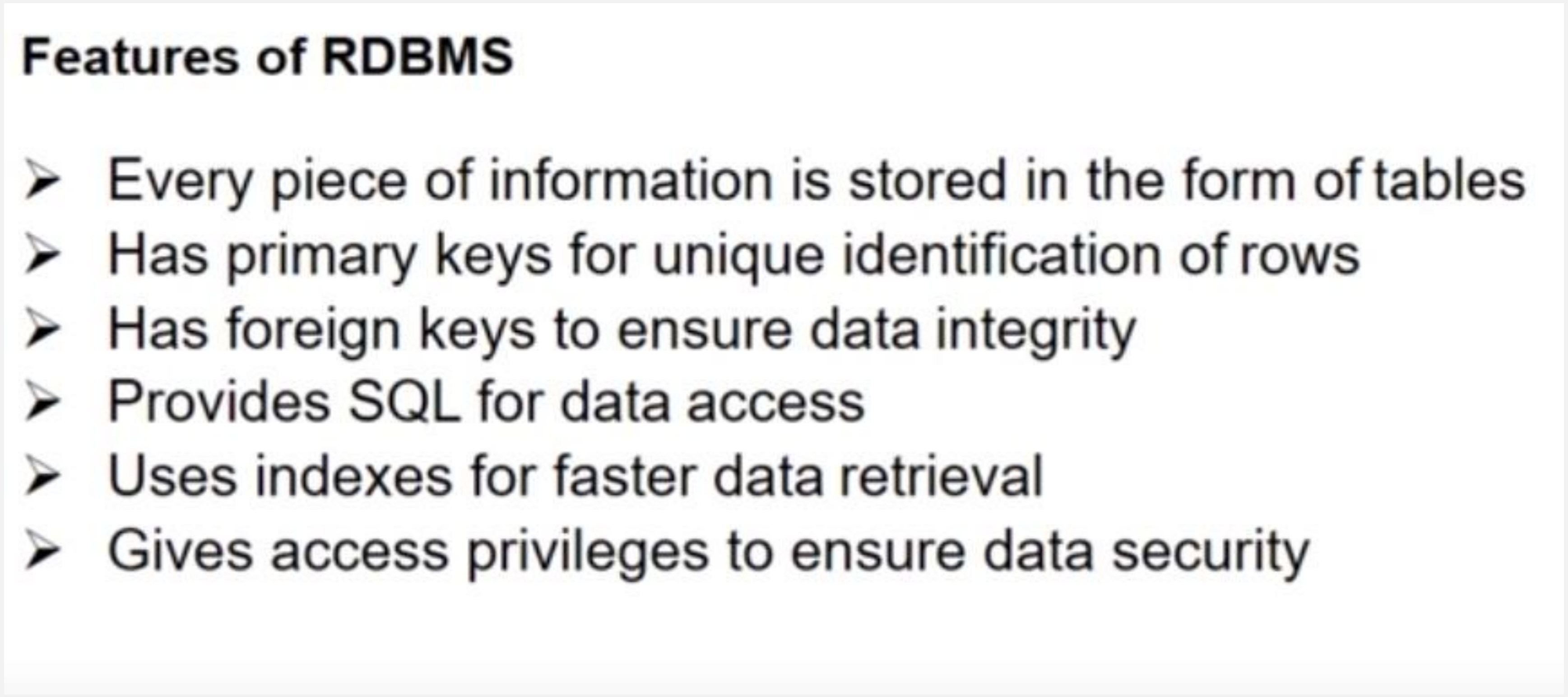


RDBMS



- A relational database refers to a database that stores data in a structured format, using rows and columns.
- This makes it easier to locate and access specific values within the database.
- It is "relational" because the values within each table are related to each other. Tables may also be related to other tables.
- The relational structure makes it possible to run queries across multiple tables at once.

RDBMS Features



Features of RDBMS

- Every piece of information is stored in the form of tables
- Has primary keys for unique identification of rows
- Has foreign keys to ensure data integrity
- Provides SQL for data access
- Uses indexes for faster data retrieval
- Gives access privileges to ensure data security

RDBMS Software's

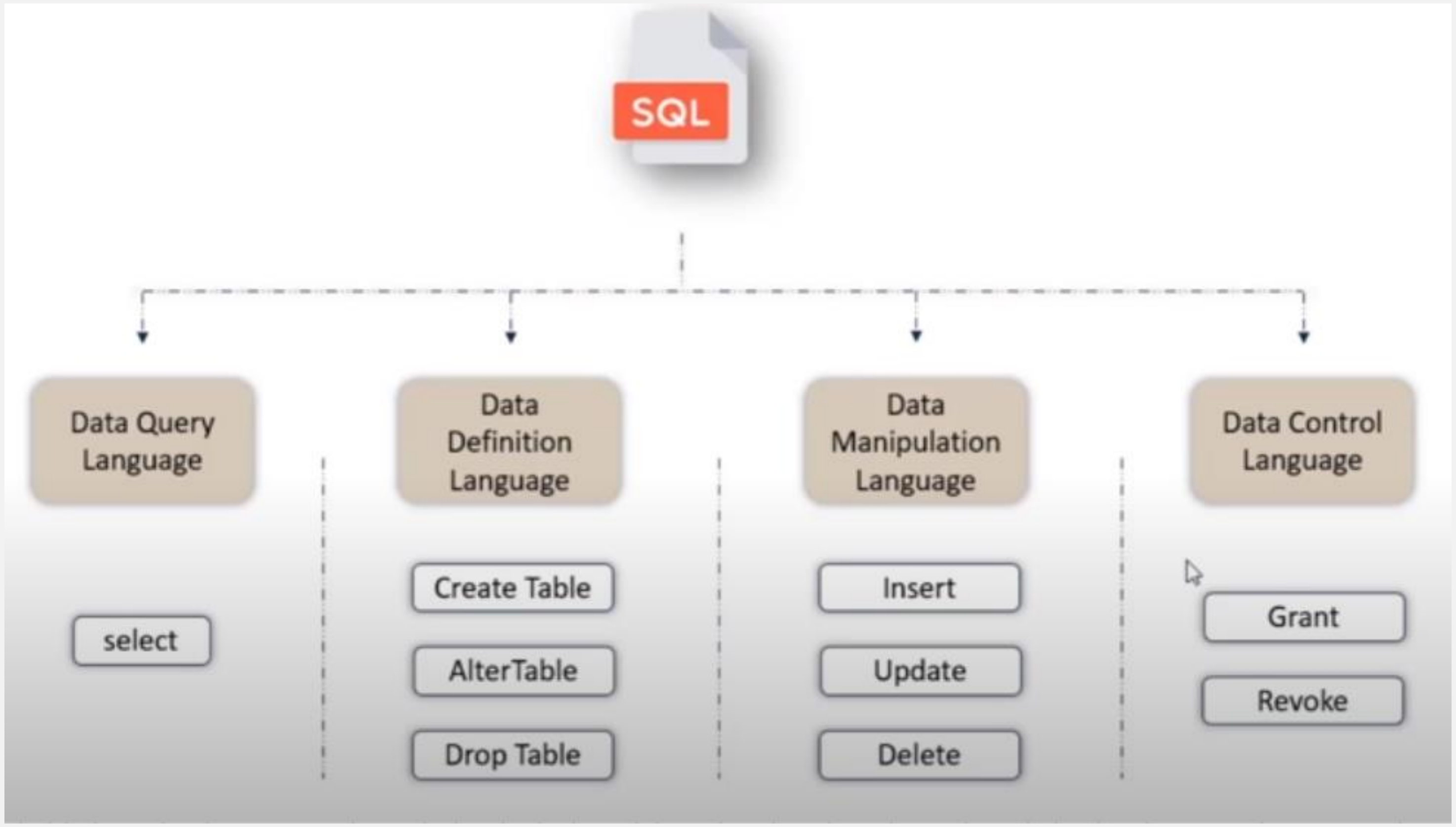
MySQL
ORACLE
Microsoft SQL Server
SYBASE

Introduction to SQL

SQL stands for Structured Query Language which is a standard language for accessing & manipulating databases

SQL is pronounced as "See-Quel"

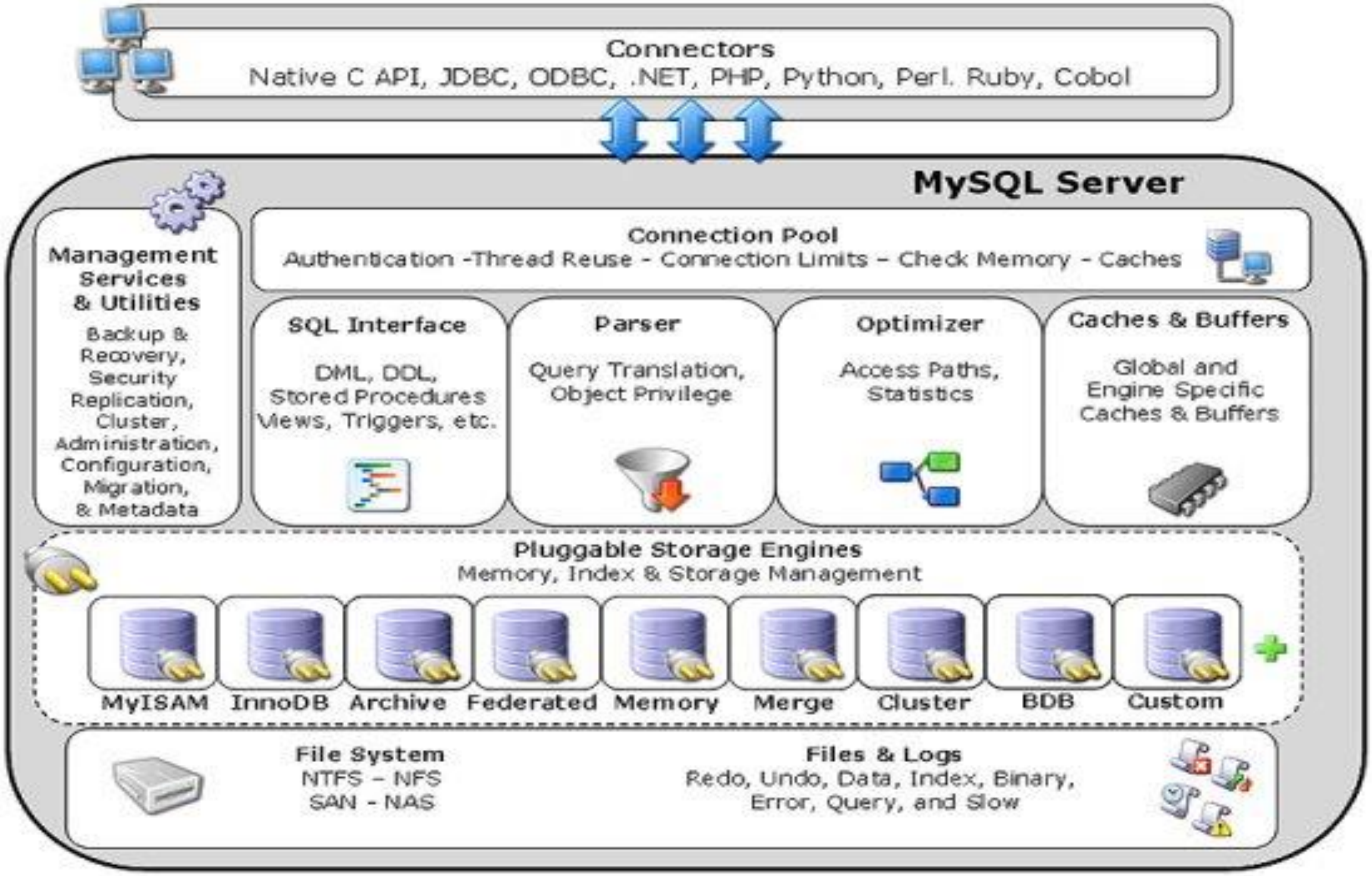
SQL Command Categories



MySQL

- MySQL is a widely used relational database management system (RDBMS).
- MySQL is free and open-source.
- MySQL is ideal for both small and large applications.
- MySQL is cross platform which means it runs on a number of different platforms such as Windows, Linux, and Mac OS etc.

MySQL Engines


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl. Ruby, Cobol
MySQL Server
Management Services & Utilities
Backup & Recovery, Security Replication, Cluster, Administration, Configuration, Migration, & Metadata
Connection Pool
Authentication -Thread Reuse - Connection Limits - Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Archive
Federated
Memory
Merge
Cluster
BDB
Custom
File System
NTFS - NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

Why MySQL ?

MySQL supports multiple storage engines each with its own specifications while other systems like SQL server only support a single storage engine.

InnoDB: – its default storage engine provided with MySQL as of version 5.5. InnoDB supports foreign keys for referential integrity and also supports ACID-standard transactions.

MyISAM: – it was the default storage engine for MySQL prior to version 5.5. MyISAM lacks support for transactions. Its advantages over InnoDB include simplicity and high performance.

Why MySQL ?

MySQL has high performance compared to other relation database systems. This is due to its simplicity in design and support for multiple-storage engines.

Cost effective, it's relatively cheaper in terms of cost when compared to other relational databases. In fact, the community edition is free. The commercial edition has a licensing fee which is also cost effective compared to licensing fees for products such as Microsoft SQL Server.

Cross platform – MySQL works on many platforms which means it can be deployed on most machines. Other systems such as MS SQL Server only run on the windows platform.

Database Normalization

- Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships.

- The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

Database Normalization

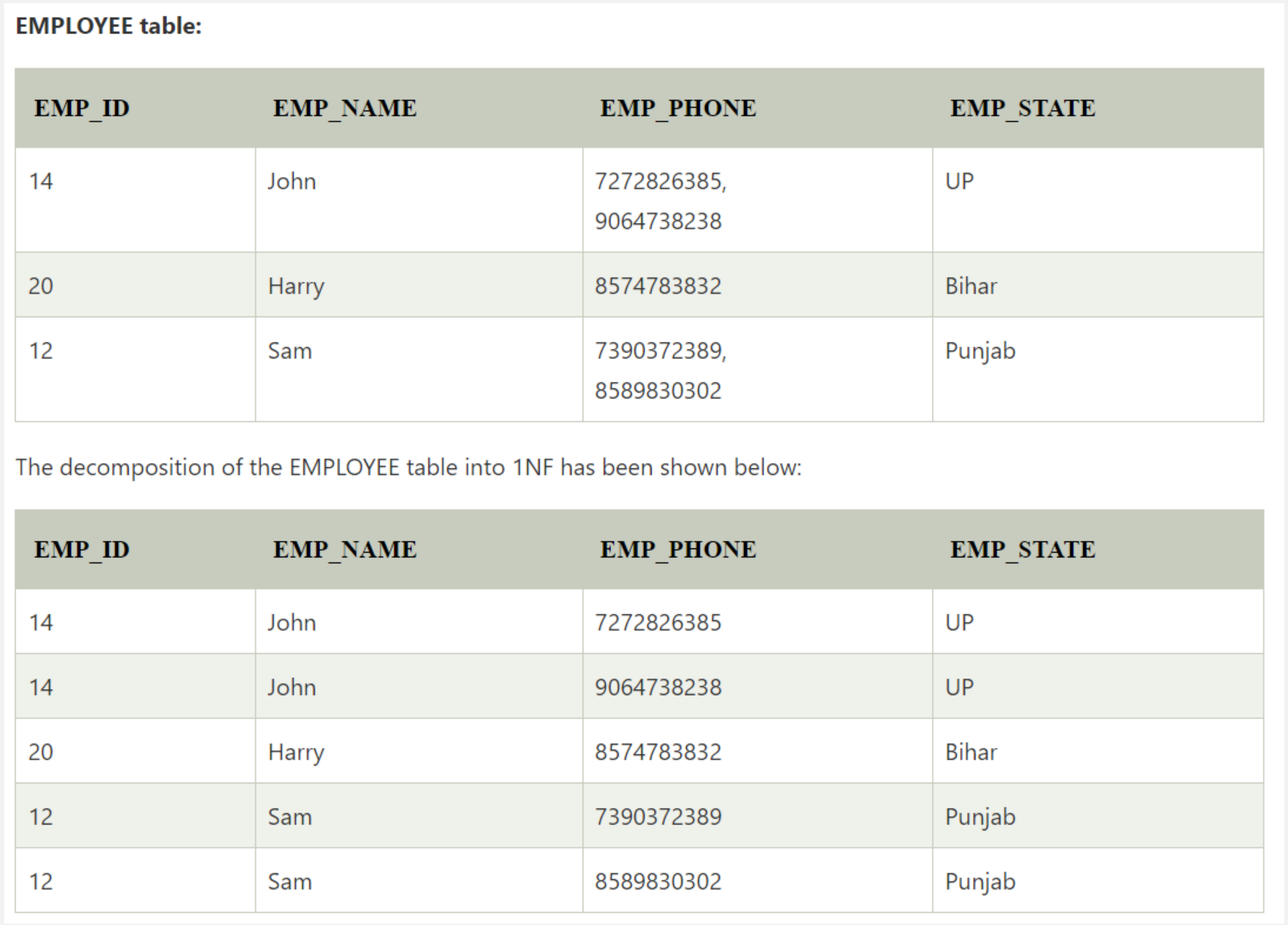
Here is a list of Normal Forms in SQL:

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
-
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form

1NF (First Normal Form) Rules

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

1NF (First Normal Form) Rules



EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

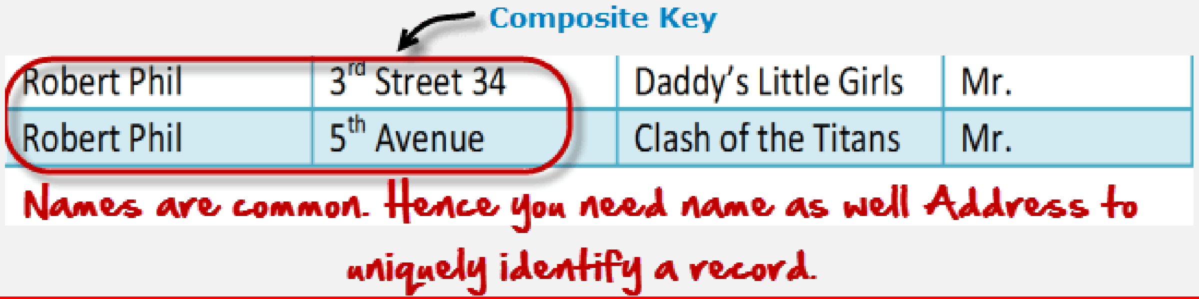
EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Primary Key in SQL

- A primary is a single column value used to identify a database record uniquely. It has following attributes

- A primary key cannot be NULL
- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

- A composite key is a primary key composed of multiple columns used to identify a record uniquely


Composite Key

Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Names are common. Hence you need name as well Address to uniquely identify a record.

2NF (Second Normal Forms) Rules

Rule 1- Be in 1NF

Rule 2- Single Column Primary Key that does not functionally dependent on any subset of candidate key relation

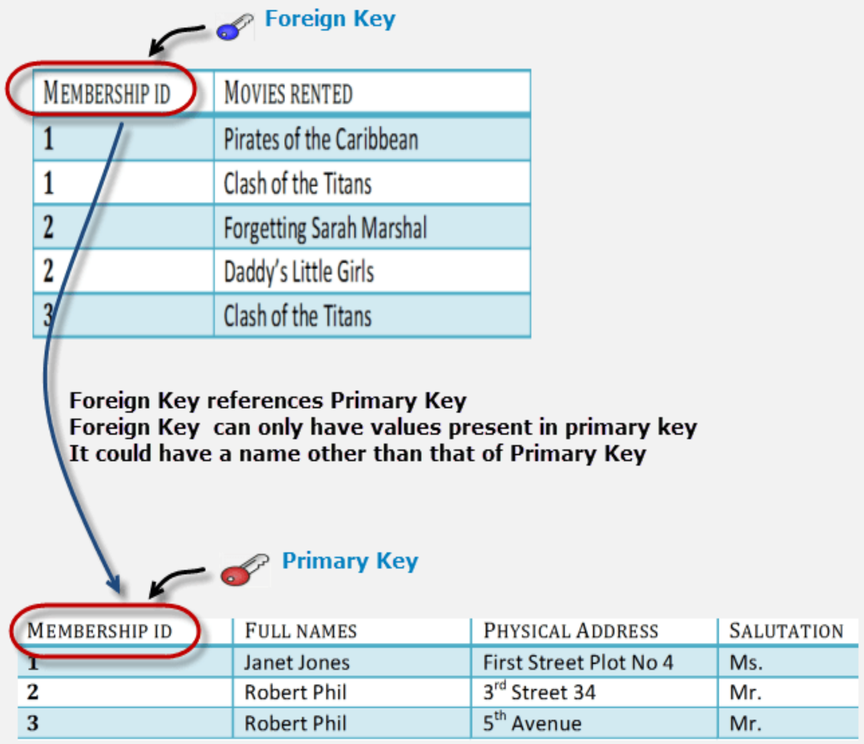
Membership ID	Full Names	Physical Address	Salutation
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Membership ID	Movies rented
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Foreign Key

Foreign Key references the primary key of another Table! It helps connect your Tables

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not



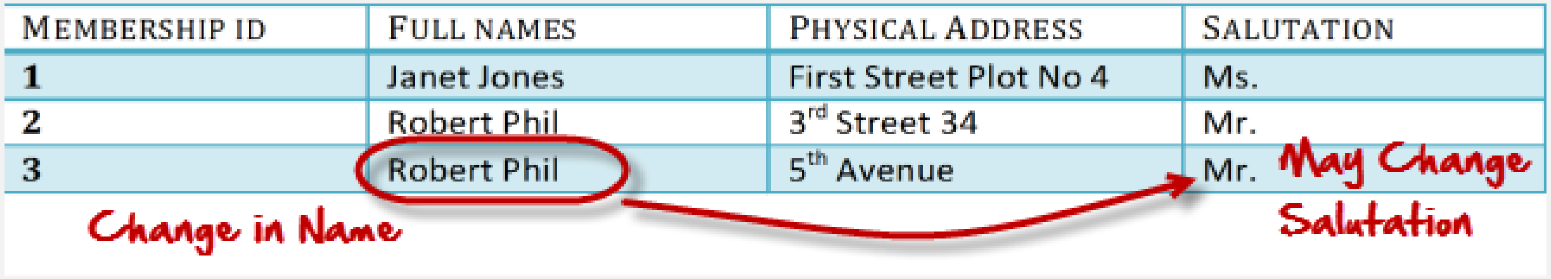
MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Foreign Key references Primary Key
Foreign Key can only have values present in primary key
It could have a name other than that of Primary Key

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Transitive Functional Dependencies

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change



MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Change in Name
May Change Salutation

3NF (Third Normal Form) Rules

Rule 1- Be in 2NF

Rule 2- Has no transitive functional dependencies

Membership ID	Full Names	Physical Address	Salutation ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

Membership ID	Movies rented
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Salutation ID	Salutation
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

MySQL Numeric Data Types

INT – A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.

TINYINT – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.

SMALLINT – A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.

MySQL Numeric Data Types

MEDIUMINT – A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.

BIGINT – A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.

MySQL Numeric Data Types

FLOAT(M,D) – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

DOUBLE(M,D) – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

DECIMAL(M,D) – An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

MySQL String Data Types

CHAR(M) – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

VARCHAR(M) – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.

BLOB or TEXT – A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

MySQL String Data Types

TINYBLOB or TINYTEXT – A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.

MEDIUMBLOB or MEDIUMTEXT – A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

LONGBLOB or LONGTEXT – A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

ENUM – An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

MySQL Date and Time Data Types

DATE – A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.

DATETIME – A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

TIMESTAMP – A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).

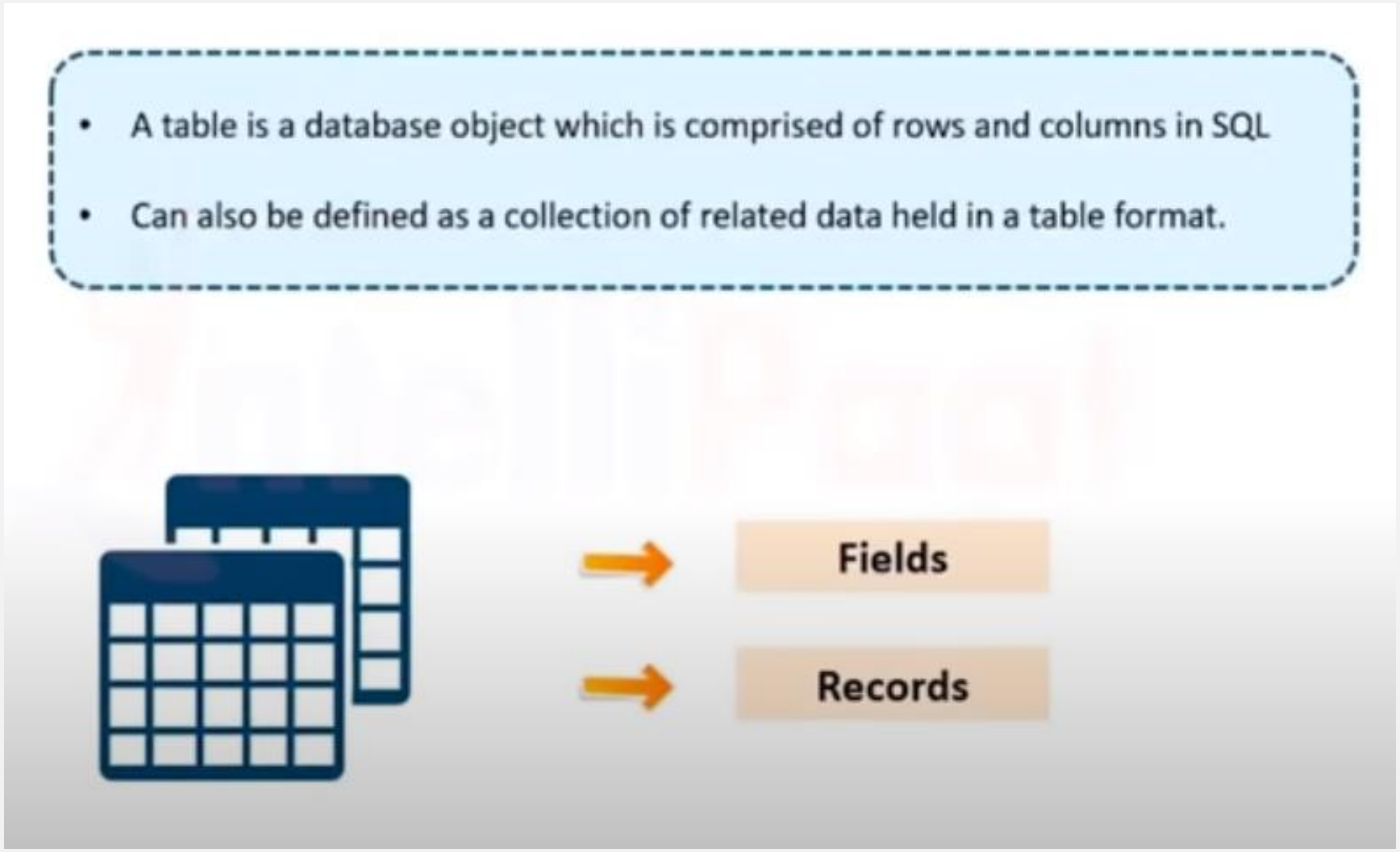
TIME – Stores the time in a HH:MM:SS format.

YEAR(M) – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

Create Database

- `CREATE DATABASE` *`databasename`*`;`
- `SHOW DATABASES`
- `DROP DATABASE` *`databasename`*`;`

MySQL Tables



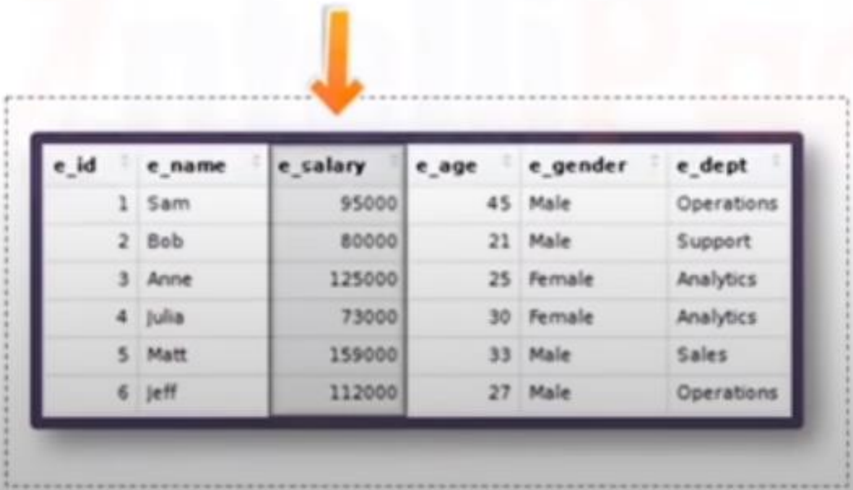
- A table is a database object which is comprised of rows and columns in SQL
- Can also be defined as a collection of related data held in a table format.

Fields
Records

SQL Table Fields

Fields are basically columns in a table with specific information about the data

Snapshot below: There is an e_salary field in the table which provides information about the salary of different employees



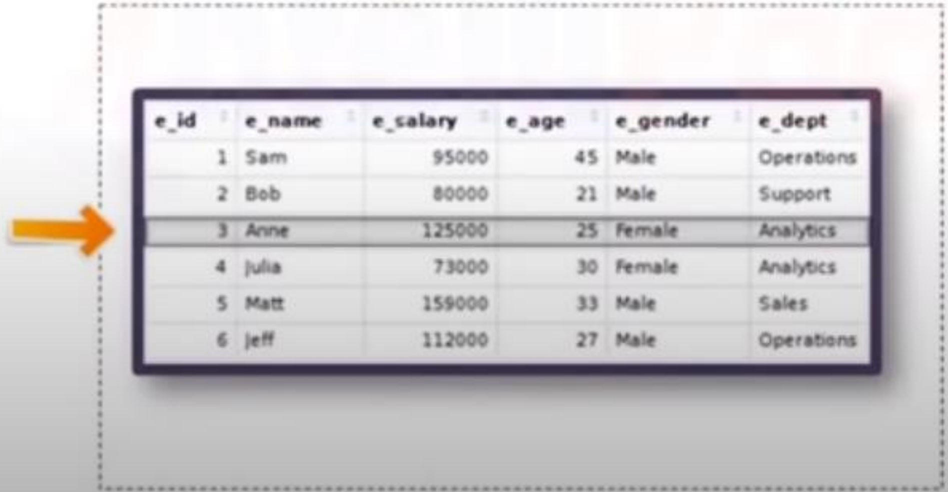
e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

SQL Table Records

A record is basically an individual entry that exists in a table

Records give the complete information of a single entry or entity.

Snapshot: One row is selected, i.e., Anne. This row gives the complete information of about the particular employee, Anne.



e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

SQL Schemas

A schema is a collection of database objects including tables, views, triggers, stored procedures, indexes, etc. A schema is associated with a username which is known as the schema owner, who is the owner of the logically related database objects.

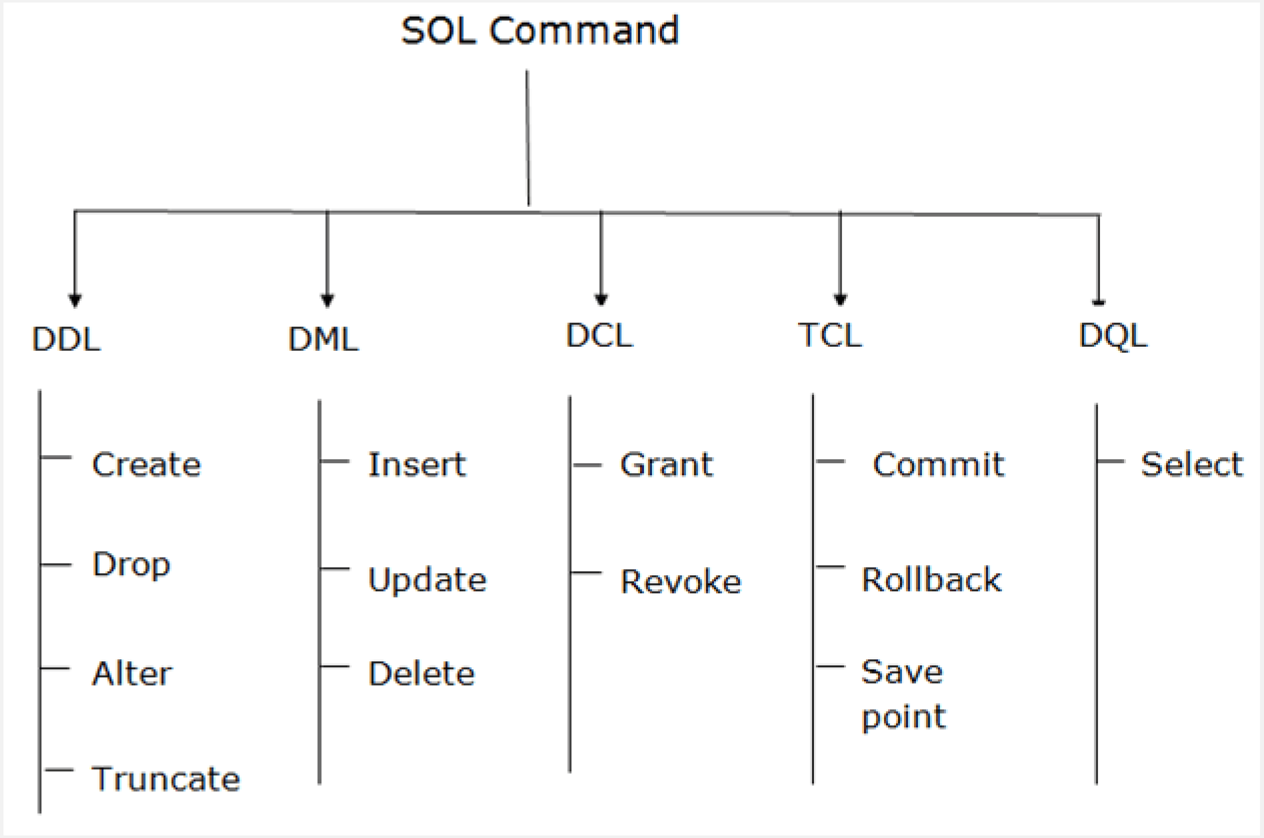
Built-in schemas in SQL Server

SQL Server provides us with some pre-defined schemas which have the same names as the built-in database users and roles, for example:

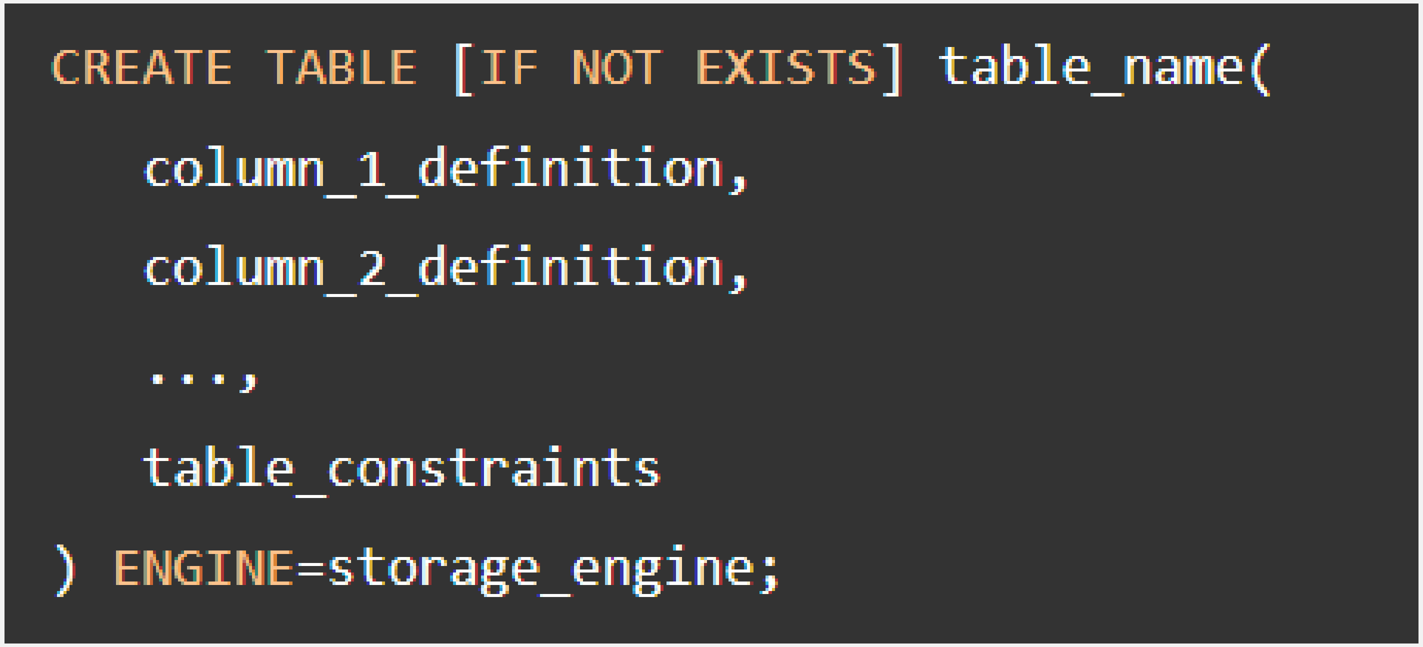
dbo, guest, sys, and INFORMATION_SCHEMA.

```
CREATE SCHEMA schema_name
```

SQL Command Types



Create Table



```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_1_definition,
    column_2_definition,
    ...,
    table_constraints
) ENGINE=storage_engine;
```

Create Table – Column Definition

```
column_name  data_type(length)

[NOT NULL]
[DEFAULT value]
[AUTO_INCREMENT]
column_constraint;
```

Constraints

The following are the most common constraints used in the SQL

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Not Null Constraint

NOT NULL Constraint

Columns in SQL Server by default store NULL values. We can restrict NULL value from being inserted into a given column by using a NOT NULL constraint.

Unique Constraint

The UNIQUE constraint ensures that no duplicate values can be inserted into a column or combination of columns that are not part of the PRIMARY KEY and are participating in the UNIQUE constraint. This constraint always inserts unique and non-repetitive values into the column. It is similar to the primary key, but it allows one null value.

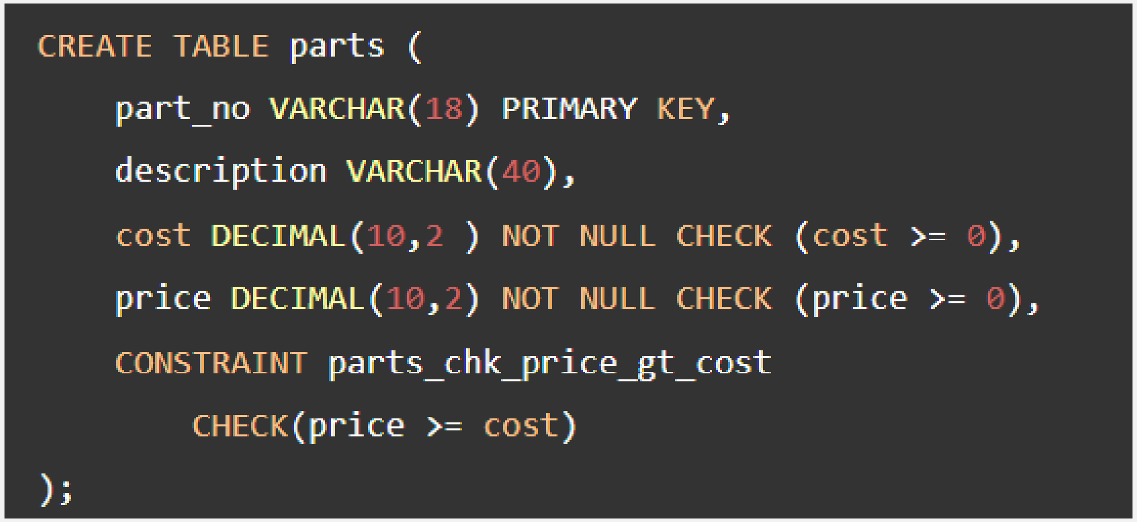
```
CREATE TABLE table_name(
    ...,
    column_name data_type UNIQUE,
    ...
);
```

```
CREATE TABLE table_name(
   ...
   column_name1 column_definition,
   column_name2 column_definition,
   ...,
   UNIQUE(column_name1,column_name2)
);
```

```
[CONSTRAINT constraint_name]
UNIQUE(column_list)
```

Check Constraint

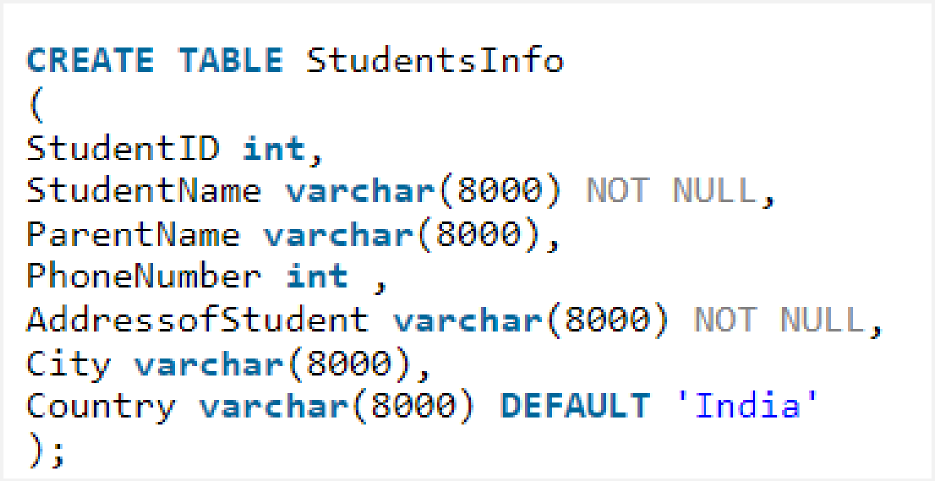
This constraint is used to limit the range of values in a column. It ensures that all the inserted values in a column must follow the specific rule.



```
CREATE TABLE parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
    CONSTRAINT parts_chk_price_gt_cost
        CHECK(price >= cost)
);
```

Default Constraint

This constraint is used to insert the default value in the column when the user does not specify any value for that column. It helps to maintain domain integrity when no value is provided into the specified default constraint column



```
CREATE TABLE StudentsInfo
(
StudentID int,
StudentName varchar(8000) NOT NULL,
ParentName varchar(8000),
PhoneNumber int ,
AddressofStudent varchar(8000) NOT NULL,
City varchar(8000),
Country varchar(8000) DEFAULT 'India'
);
```

Primary Key Constraint

- A primary key is a column or a set of columns that uniquely identifies each row in the table.

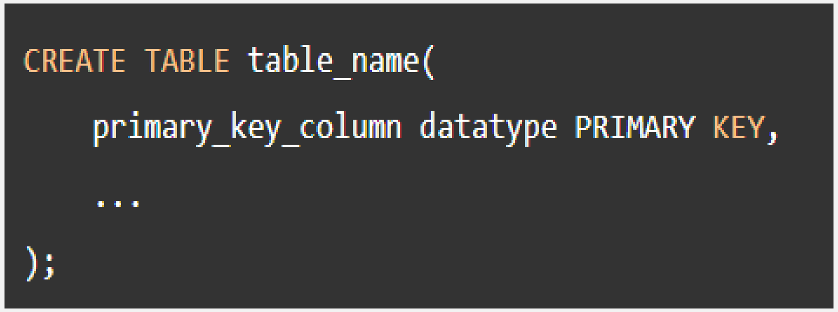
The primary key follows these rules:

  - A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.

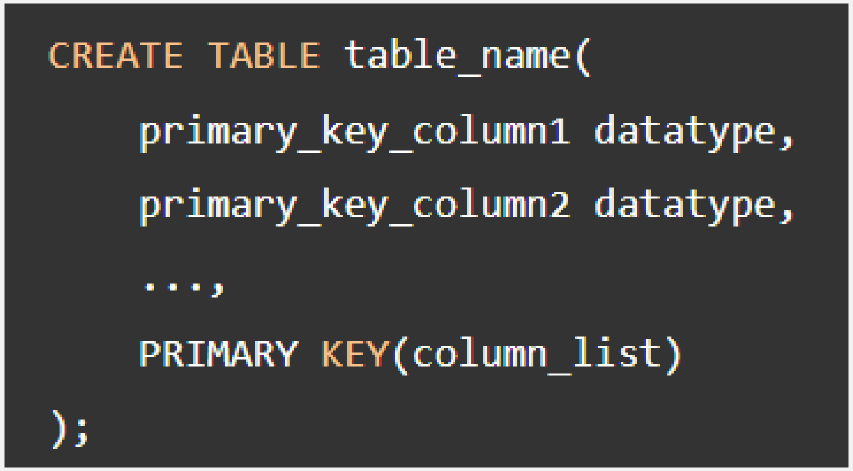
  - A primary key column cannot have NULL values. Any attempt to insert or update NULL to primary key columns will result in an error. Note that MySQL implicitly adds a NOT NULL constraint to primary key columns

  - A table can have one an only one primary key.

Primary Key Constraint



```
CREATE TABLE table_name(
    primary_key_column datatype PRIMARY KEY,
    ...
);
```



```
CREATE TABLE table_name(
    primary_key_column1 datatype,
    primary_key_column2 datatype,
    ...,
    PRIMARY KEY(column_list)
);
```

Foreign Key Constraint

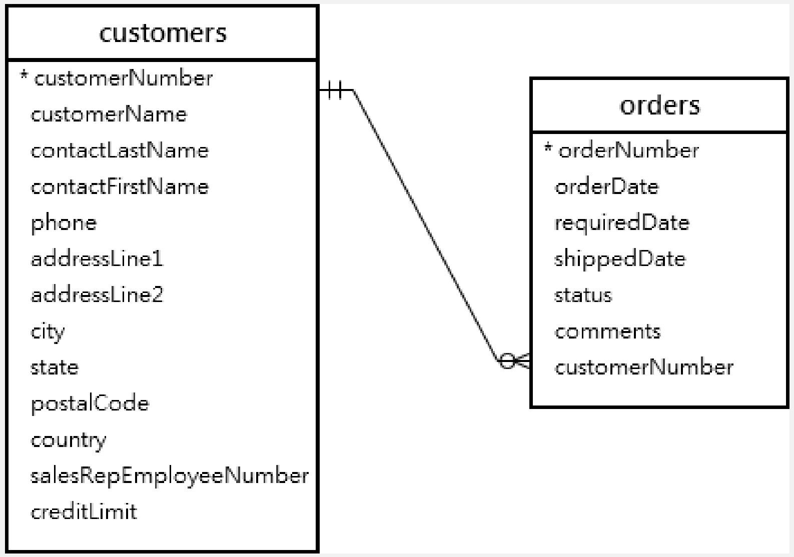
A foreign key is a database key that links two tables together. This constraint is also known as referencing key as it identifies the relationships between the tables by referencing a column of the child table containing the foreign key to the PRIMARY KEY column of the parent table. It means the foreign key column in one table refers to the primary key column of another table.

Foreign Key Constraint

The Rules of Foreign Key are as follows:

- The table with the foreign key is called the child table and the table being referenced by the foreign key is called the parent table.
- Null values are allowed in a foreign key
- Foreign keys can be duplicated
- There can be more than a single foreign key in a table
- The relationship established between the tables is known as referential integrity

FOREIGN KEY



```
[CONSTRAINT constraint_name]
FOREIGN KEY [foreign_key_name] (column_name, ...)
REFERENCES parent_table(colunm_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```

FOREIGN KEY References

CASCADE: if a row from the parent table is deleted or updated, the values of the matching rows in the child table automatically deleted or updated.

SET NULL: if a row from the parent table is deleted or updated, the values of the foreign key column (or columns) in the child table are set to NULL.

RESTRICT: if a row from the parent table has a matching row in the child table, MySQL rejects deleting or updating rows in the parent table.

NO ACTION: is the same as RESTRICT.

If you don't specify the ON DELETE and ON UPDATE clause, the default action is RESTRICT.

FOREIGN KEY Example

```
CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT NOT NULL,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
    REFERENCES categories(categoryId)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) ENGINE=INNODB;
```

Create Index

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Create Index Examples

Creates an index on a table. Duplicate values are allowed:

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

Creates a unique index on a table. Duplicate values are not allowed:

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

The DROP INDEX statement is used to delete an index in a table

Auto Increment Field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Alter Table – Add/Modify a Column/s

```
ALTER TABLE table_name
ADD
    new_column_name column_definition
    [FIRST | AFTER column_name]
```

```
ALTER TABLE table_name
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ...;
```

Alter Table – Rename/Drop Column

```
ALTER TABLE table_name
    CHANGE COLUMN original_name new_name column_definition
    [FIRST | AFTER column_name];
```

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Drop Table

```
DROP TABLE table_name;


TRUNCATE TABLE table_name;
```

Insert Into

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name (column_list)
VALUES
        (value_list_1),
        (value_list_2),
        ...
        (value_list_n);
```

Insert Into

INSERT with SELECT statement:

```
INSERT INTO table_name(column_list)
SELECT
   select_list
FROM
   another_table
WHERE
   condition;
```

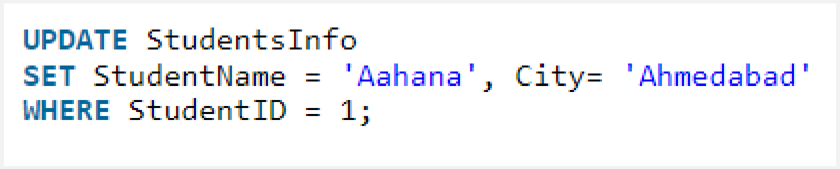
INSERT ON DUPLICATE KEY UPDATE statement

```
INSERT INTO table (column_list)
VALUES (value_list)
ON DUPLICATE KEY UPDATE
   c1 = v1,
   c2 = v2,
   ...;
```

Update

The UPDATE statement is used to modify or update the records already present in the table.

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2, ...
WHERE Condition;
```

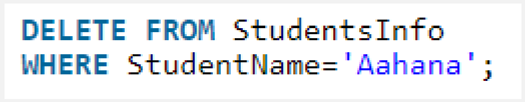


```
UPDATE StudentsInfo
SET StudentName = 'Aahana', City= 'Ahmedabad'
WHERE StudentID = 1;
```

Delete

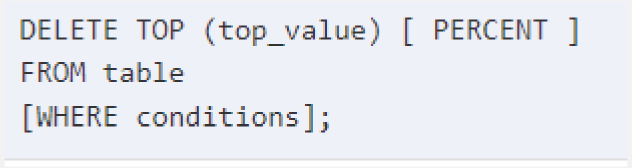
The DELETE statement is used to delete the existing records in a table.

DELETE FROM TableName WHERE Condition;

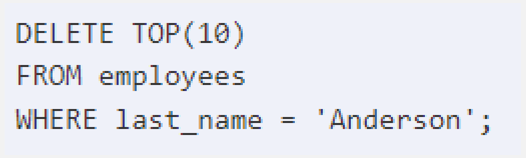


```
DELETE FROM StudentsInfo
WHERE StudentName='Aahana';
```

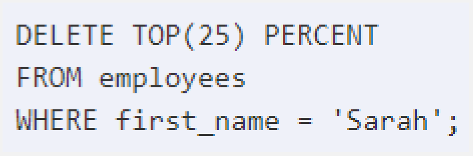
DELETE TOP Statement



```
DELETE TOP (top_value) [ PERCENT ]
FROM table
[WHERE conditions];
```



```
DELETE TOP(10)
FROM employees
WHERE last_name = 'Anderson';
```

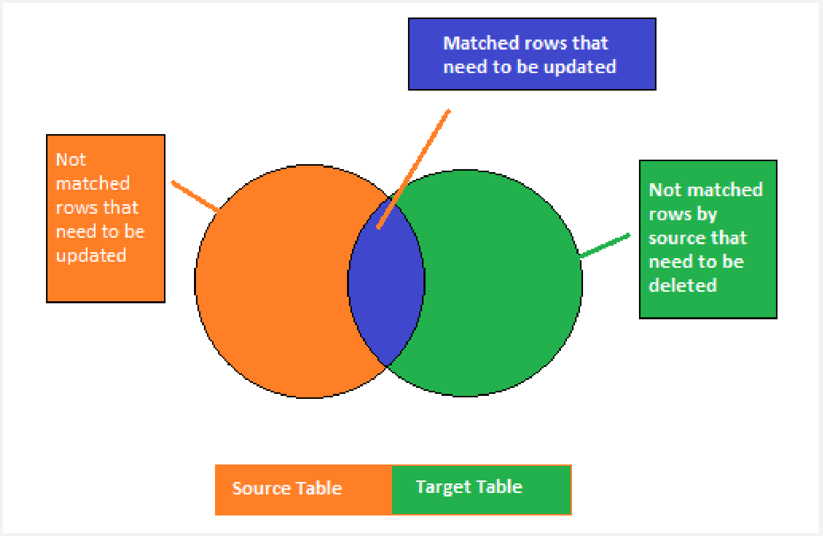


```
DELETE TOP(25) PERCENT
FROM employees
WHERE first_name = 'Sarah';
```

Merge Statement

MERGE statement combines INSERT, UPDATE, and DELETE operations into a single statement

- Some rows in the "source table" are not found in the "target table." In this situation, it is required to add rows from the source to the target table.
- Some rows in the "target table" are not found in the "source table." In this situation, it is required to delete rows from the target table.
- Some rows in the "source and target table" have the same keys. However, they have distinct values in the non-key columns. In this case, it's necessary to update the rows in the "target table" with data from the "source table."

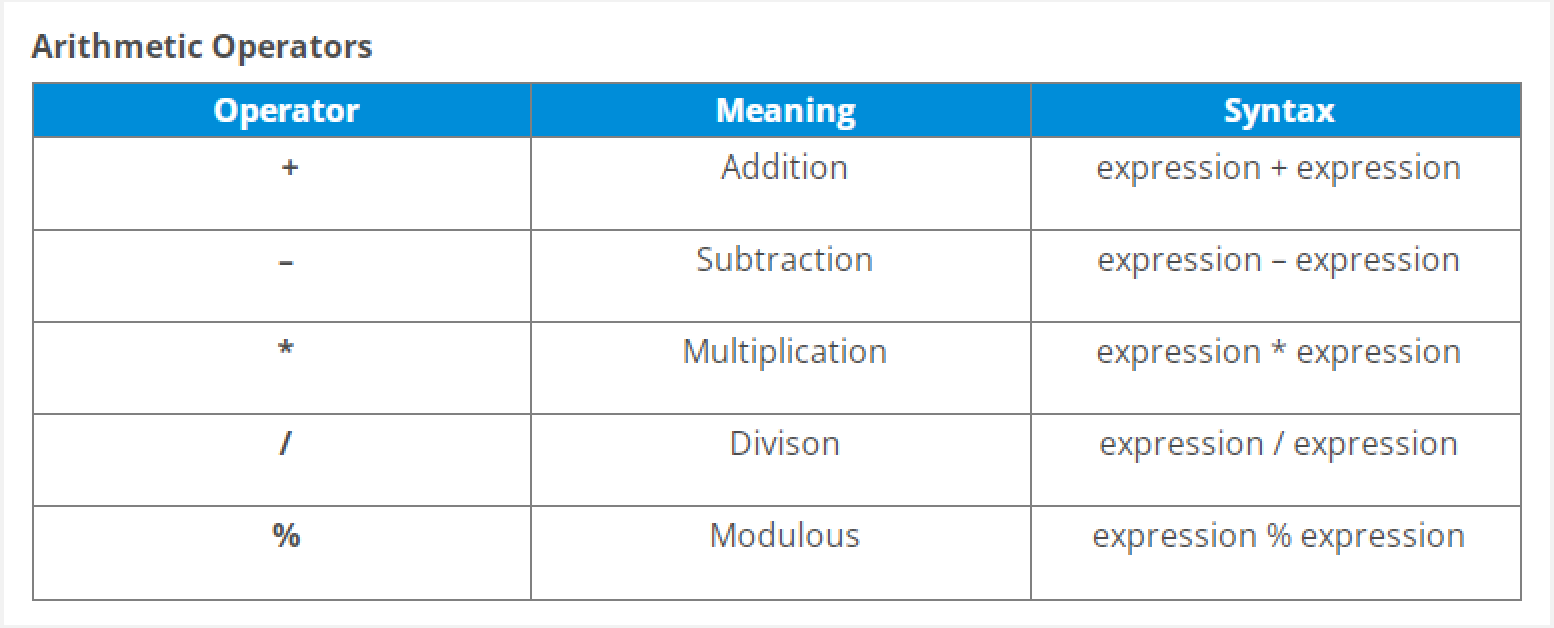

Matched rows that need to be updated
Not matched rows that need to be updated
Not matched rows by source that need to be deleted
Source Table
Target Table

Merge Statement

The following are the syntax that illustrates the MERGE statement in SQL Server:

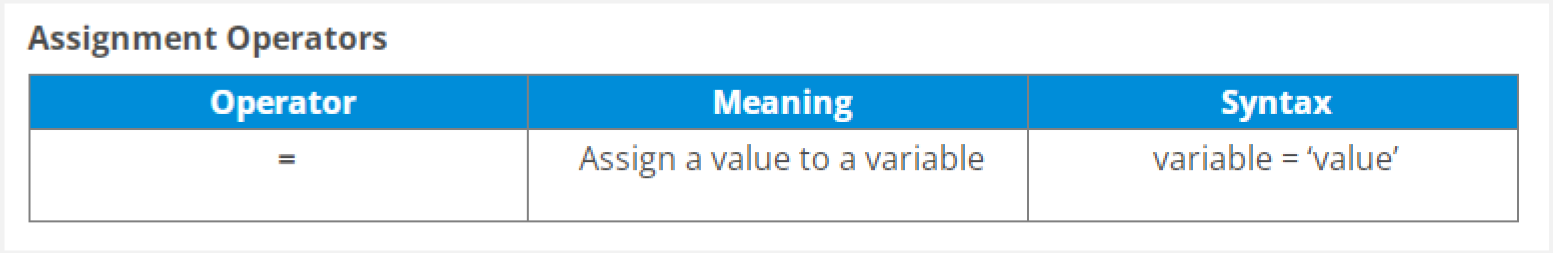
```
MERGE target_table USING source_table
ON merge_condition
WHEN MATCHED
   THEN update_statement
WHEN NOT MATCHED
   THEN insert_statement
WHEN NOT MATCHED BY SOURCE
   THEN DELETE;
```

Arithmetic Operators



Arithmetic Operators

Operator	Meaning	Syntax
+	Addition	expression + expression
–	Subtraction	expression – expression
*	Multiplication	expression * expression
/	Divison	expression / expression
%	Modulous	expression % expression



Assignment Operators

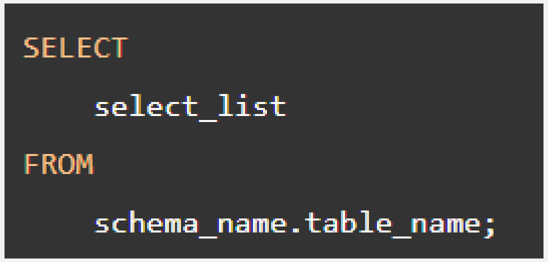
Operator	Meaning	Syntax
=	Assign a value to a variable	variable = 'value'

Bitwise Operators

Operator	Meaning	Syntax
& (Bitwise AND)	Used to perform a bitwise logical AND operation between two integer values.	expression & expression
&= (Bitwise AND Assignment)	Used to perform a bitwise logical AND operation between two integer values. It also sets a value to the output of the operation.	expression &= expression
| (Bitwise OR)	Used to perform a bitwise logical OR operation between two integer values as translated to binary expressions within Transact-SQL statements.	expression | expression
|= (Bitwise OR Assignment)	Used to perform a bitwise logical OR operation between two integer values as translated to binary expressions within Transact-SQL statements. It also sets a value to the output of the operation.	expression |= expression
^ (Bitwise Exclusive OR)	Used to perform a bitwise exclusive OR operation between two integer values.	expression ^ expression
^= (Bitwise Exclusive OR Assignment)	Used to perform a bitwise exclusive OR operation between two integer values. It also sets a value to the output of the operation.	expression ^= expression
~ (Bitwise NOT)	Used to perform a bitwise logical NOT operation on an integer value.	~ expression

SQL Select Query

The SELECT statement is used to select data from a database, table or view.



```
SELECT
    select_list
FROM
    schema_name.table_name;
```

SQL Select Query – Order By

ORDER BY

This statement is used to sort the required results either in the ascending or descending order. By default, the results are stored in ascending order. Yet, if you wish to get the results in descending order, you have to use the DESC keyword.

Syntax

```
SELECT Column1, Column2, ...ColumnN
FROM TableName
ORDER BY Column1, Column2, ... ASC|DESC;
```

SQL Select Query – Offset

OFFSET AND FETCH

The OFFSET and FETCH clauses are the options of the ORDER BY clause. They allow you to limit the number of rows to be returned by a query.

Syntax

```
ORDER BY column_list [ASC |DESC]
OFFSET offset_row_count {ROW | ROWS}
FETCH {FIRST | NEXT} fetch_row_count {ROW |
ROWS} ONLY
```

SQL Select Query – Select Top

The SELECT TOP clause allows you to limit the number of rows or percentage of rows returned in a query result set.

```
SELECT TOP (expression) [PERCENT]
    [WITH TIES]
FROM
    table_name
ORDER BY
    column_name;
```

SQL Select Query – Distinct

DISTINCT

The DISTINCT keyword is used with the SELECT statement to return only different values.

Syntax

```
SELECT DISTINCT Column1, Column2, ...ColumnN
FROM TableName;
```

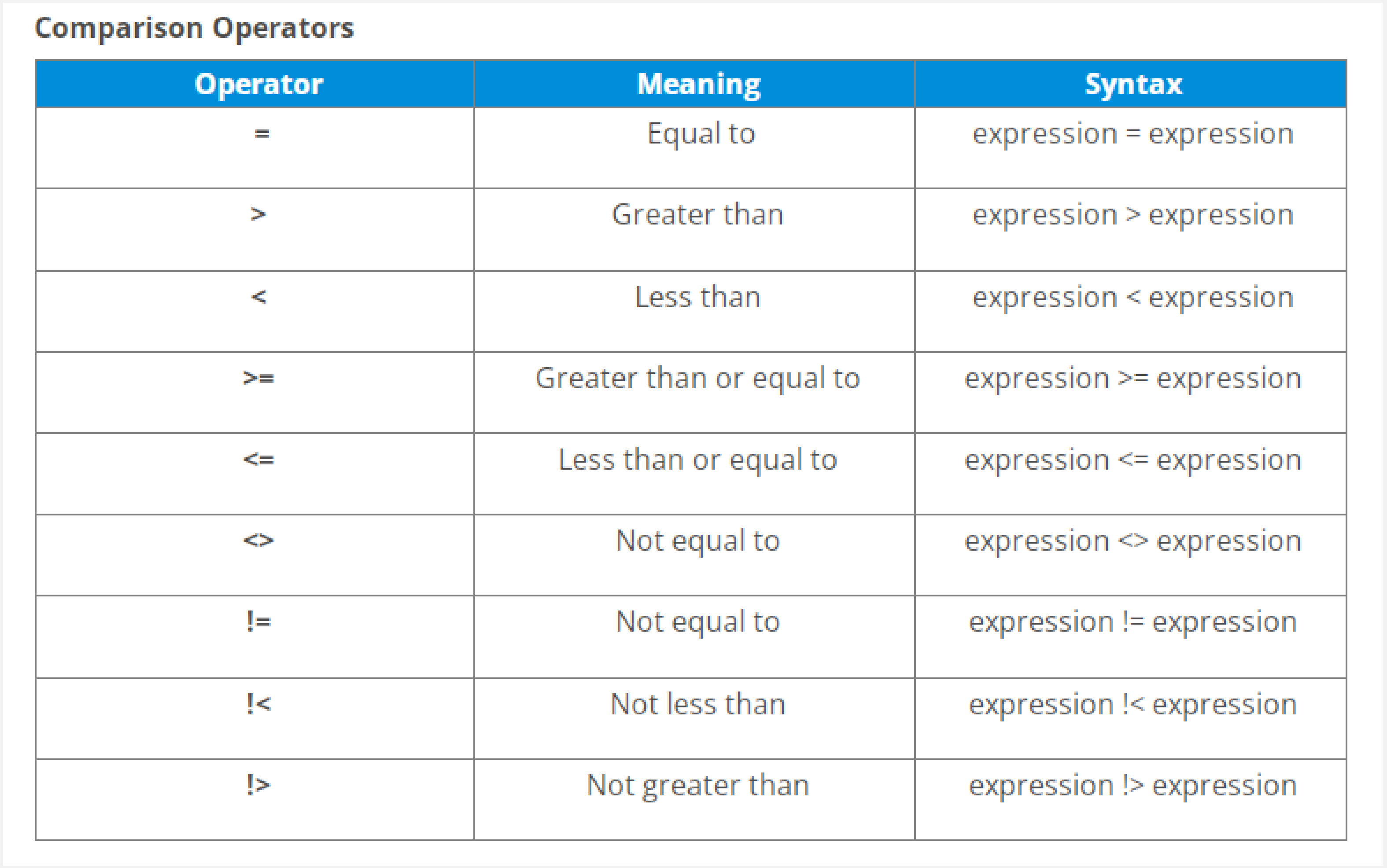
SQL Select Query – Where

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Relational Operators



Comparison Operators

Operator	Meaning	Syntax
=	Equal to	expression = expression
>	Greater than	expression > expression
<	Less than	expression < expression
>=	Greater than or equal to	expression >= expression
<=	Less than or equal to	expression <= expression
<>	Not equal to	expression <> expression
!=	Not equal to	expression != expression
!<	Not less than	expression !< expression
!>	Not greater than	expression !> expression

Logical Operators

Operator	Meaning	Syntax
ALL	Returns TRUE if all of set of comparisons are TRUE.	scalar_expression { = | <> | != | > | >= | !> | < | <= | !< } ALL (subquery)
AND	Returns TRUE if both the expressions are TRUE.	boolean_expression AND boolean_expression
ANY	Returns TRUE if any one of a set of comparisons are TRUE.	scalar_expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } { ANY } (subquery)
BETWEEN	Returns TRUE if an operand is within a range.	sampleexpression [NOT] BETWEEN beginexpression AND endexpression
EXISTS	Returns TRUE if a subquery contains any rows.	EXISTS (sub query)
IN	Returns TRUE if an operand is equal to one of a list of expressions.	test_expression [NOT] IN(subquery | expression [,...n])
LIKE	Returns TRUE if an operand matches a pattern.	match_expression [NOT] LIKE pattern [ESCAPE escape_character]
NOT	Reverses the value of any boolean operator.	[NOT] boolean_expression
OR	Returns TRUE if either of the boolean expression is TRUE.	boolean_expression OR boolean_expression
SOME	Returns TRUE if some of a set of comparisons are TRUE.	scalar_expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } { SOME} (subquery)

Like Operator

Wildcard Characters in SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

Like Operator

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Aggregate Functions

Function	Description	Syntax	Example
SUM()	Used to return the sum of a group of values.	SELECT SUM(ColumnName) FROM TableName;	SELECT SUM(Marks) FROM StudentsInfo;
COUNT()	Returns the number of rows either based on a condition, or without a condition.	SELECT COUNT(ColumnName) FROM TableName WHERE Condition;	SELECT COUNT(StudentID) FROM StudentsInfo;
AVG()	Used to calculate the average value of a numeric column.	SELECT AVG(ColumnName) FROM TableName;	SELECT AVG(Marks) FROM StudentsInfo;
MIN()	This function returns the minimum value of a column.	SELECT MIN(ColumnName) FROM TableName;	SELECT MIN(Marks) FROM StudentsInfo;
MAX()	Returns a maximum value of a column.	SELECT MAX(ColumnName) FROM TableName;	SELECT MAX(Marks) FROM StudentsInfo;
FIRST()	Used to return the first value of the column.	SELECT FIRST(ColumnName) FROM TableName;	SELECT FIRST(Marks) FROM StudentsInfo;
LAST()	This function returns the last value of the column.	SELECT LAST(ColumnName) FROM TableName;	SELECT LAST(Marks) FROM StudentsInfo;

Joins

Joins are used to combine tuples from two or more tables, based on a related column between the tables.

There are four types of joins:

INNER JOIN: Returns records that have matching values in both the tables.

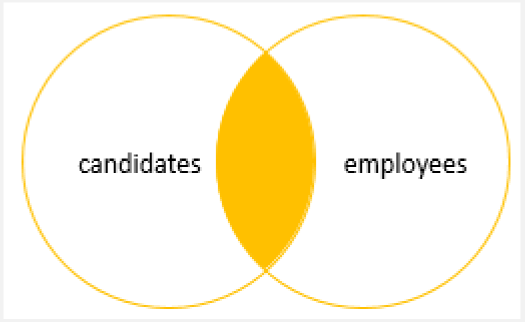
LEFT JOIN: Returns records from the left table, and also those records which satisfy the condition from the right table.

RIGHT JOIN: Returns records from the right table, and also those records which satisfy the condition from the left table.

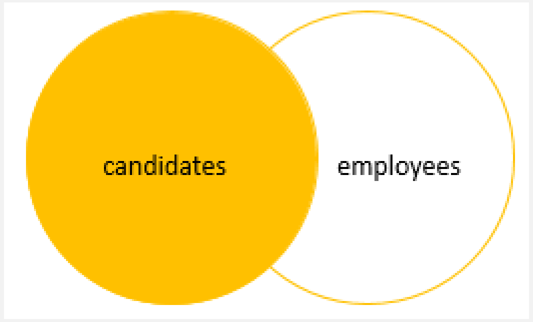
FULL JOIN: Returns records which either have a match in the left or the right table.

Joins

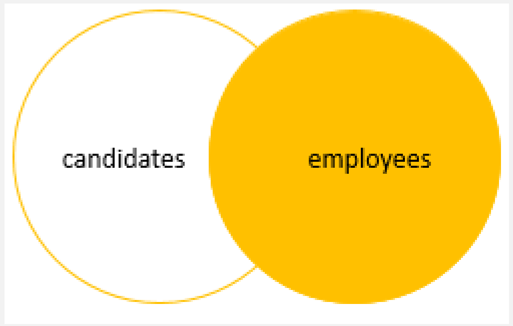
SQL Server Inner Join



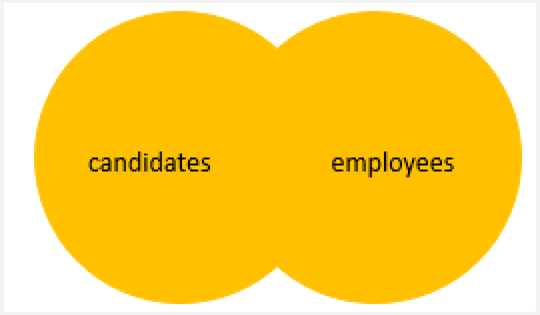
SQL Server Left Join



SQL Server Right Join



SQL Server full join



Group by

The GROUP BY clause allows you to arrange the rows of a query in groups. The groups are determined by the columns that you specify in the GROUP BY clause.

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    column_name1,
    column_name2 ,...;
```

Having

The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified list of conditions. The following illustrates the HAVING clause syntax:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    group_list
HAVING
    conditions;
```

```
SELECT
    column_name1,
    column_name2,
    aggregate_function (column_name3) alias
FROM
    table_name
GROUP BY
    column_name1,
    column_name2
HAVING
    aggregate_function (column_name3) >
value;
```

Cube

CUBE is an extension of the GROUP BY clause. It allows you to generate the sub-totals for all the combinations of the grouping columns specified in the GROUP BY clause.

Syntax

```
SELECT ColumnName(s)
FROM TableName
GROUP BY CUBE(ColumnName1,
ColumnName2, ....., ColumnNameN);
```

Example

```
SELECT StudentID, COUNT(City)
FROM StudentsInfo
GROUP BY CUBE(StudentID)
ORDER BY StudentID;
```

Subquery

A subquery is a query nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE.

```
SELECT    order_id,    order_date,    customer_id
FROM
   sales.orders
WHERE
   customer_id IN (
      SELECT
         customer_id
      FROM
         sales.customers
      WHERE
         city = 'New York'
   )
ORDER BY
   order_date DESC;
```

Subquery Exists

The EXISTS operator is a logical operator that allows you to check whether a subquery returns any row. The EXISTS operator returns TRUE if the subquery returns one or more rows.

```
EXISTS ( subquery)
```

Any Operator

The ANY operator is a logical operator that compares a scalar value with a single-column set of values returned by a subquery.

scalar_expression comparison_operator ANY (subquery)

- scalar_expression is any valid expression.
- comparison_operator is any comparison operator.
- subquery is a SELECT statement which returns a result set of a single column with the data is the same as the data type of the scalar expression.

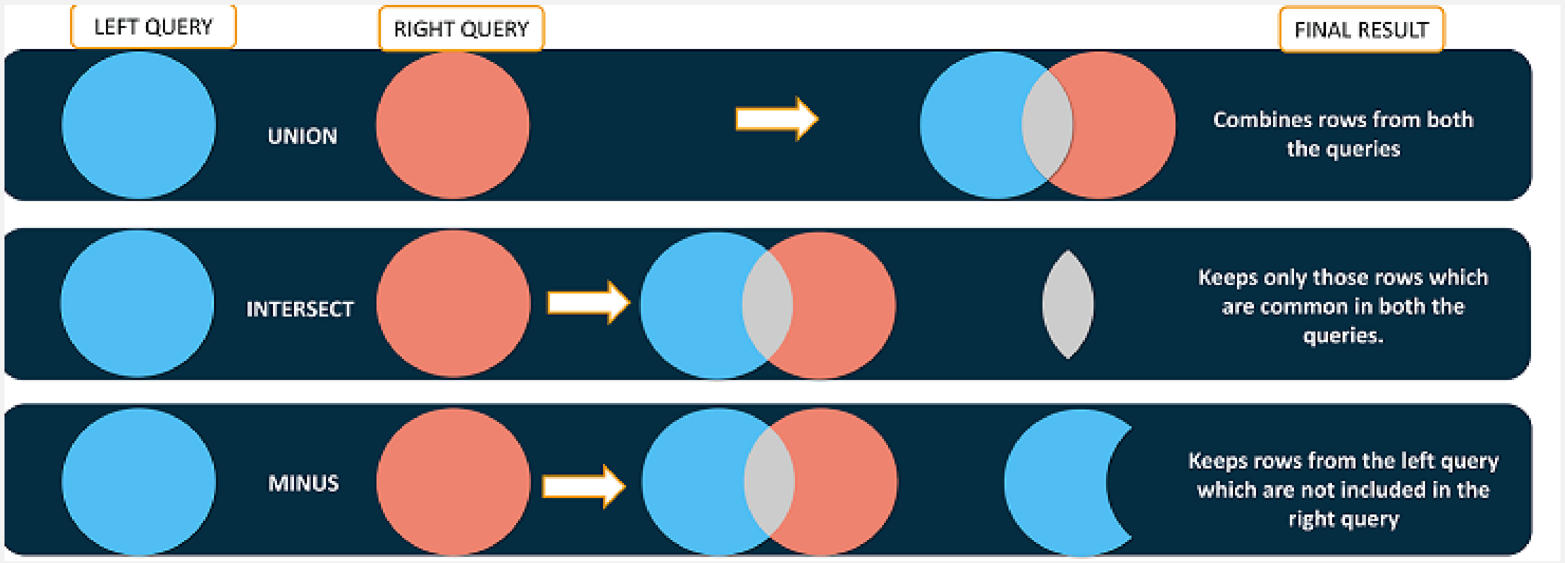
All Operator

The SQL Server ALL operator is a logical operator that compares a scalar value with a single-column list of values returned by a subquery.

scalar_expression comparison_operator ALL (subquery)

- The scalar_expression is any valid expression.
- The comparison_operator is any valid comparison operator including equal (=), not equal (<>), greater than (>), greater than or equal (>=), less than (<), less than or equal (<=).
- The subquery within the parentheses is a SELECT statement that returns a result of a single column. Also, the data type of the returned column must be the same data type as the data type of the scalar expression.

Set Operators


LEFT QUERY
RIGHT QUERY
FINAL RESULT
UNION
Combines rows from both the queries
INTERSECT
Keeps only those rows which are common in both the queries.
MINUS
Keeps rows from the left query which are not included in the right query

Set Operators

Operator	Meaning	Syntax

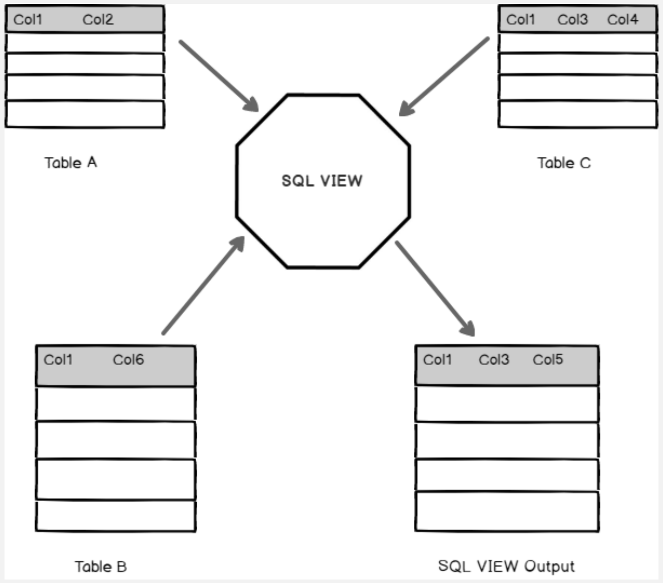
UNION	The UNION operator is used to combine the result-set of two or more SELECT statements.	SELECT ColumnName(s) FROM Table1
		UNION
		SELECT ColumnName(s)FROM Table2;

INTERSECT	The INTERSECT clause is used to combine two SELECT statements and return the intersection of the data-sets of both the SELECT statements.	SELECT Column1 , Column2 ….
		FROM TableName;
		WHERE Condition
		INTERSECT
		SELECT Column1 , Column2 ….
		FROM TableName;
		WHERE Condition

EXCEPT	The EXCEPT operator returns those tuples that are returned by the first SELECT operation, and are not returned by the second SELECT operation.	SELECT ColumnName
		FROM TableName;
		EXCEPT
		SELECT ColumnName
		FROM TableName;

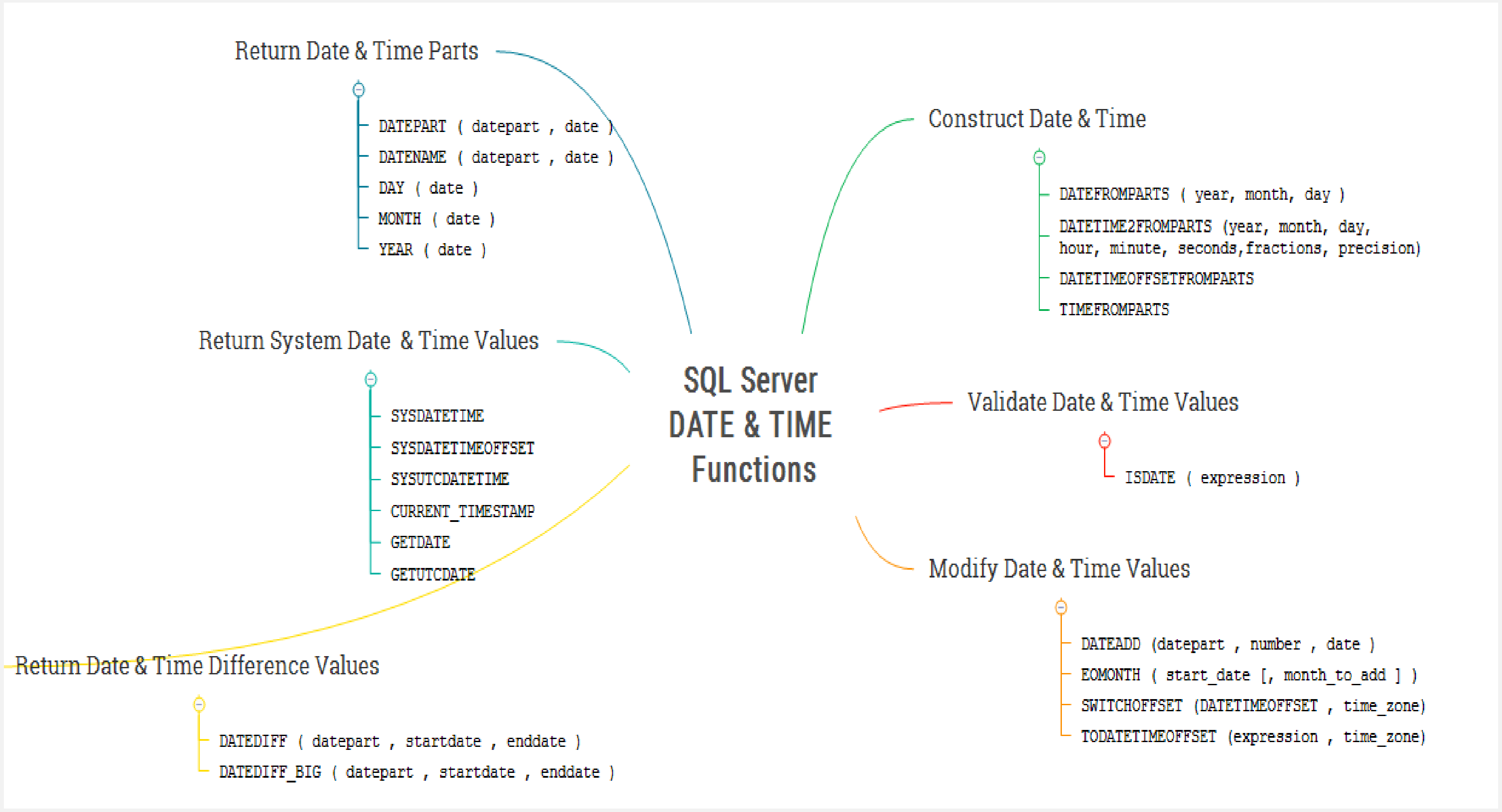
Views in SQL

A VIEW in SQL Server is like a virtual table that contains data from one or multiple tables. It does not hold any data and does not exist physically in the database. Similar to a SQL table, the view name should be unique in a database. It contains a set of predefined SQL queries to fetch data from the database. It can contain database tables from single or multiple databases as well.

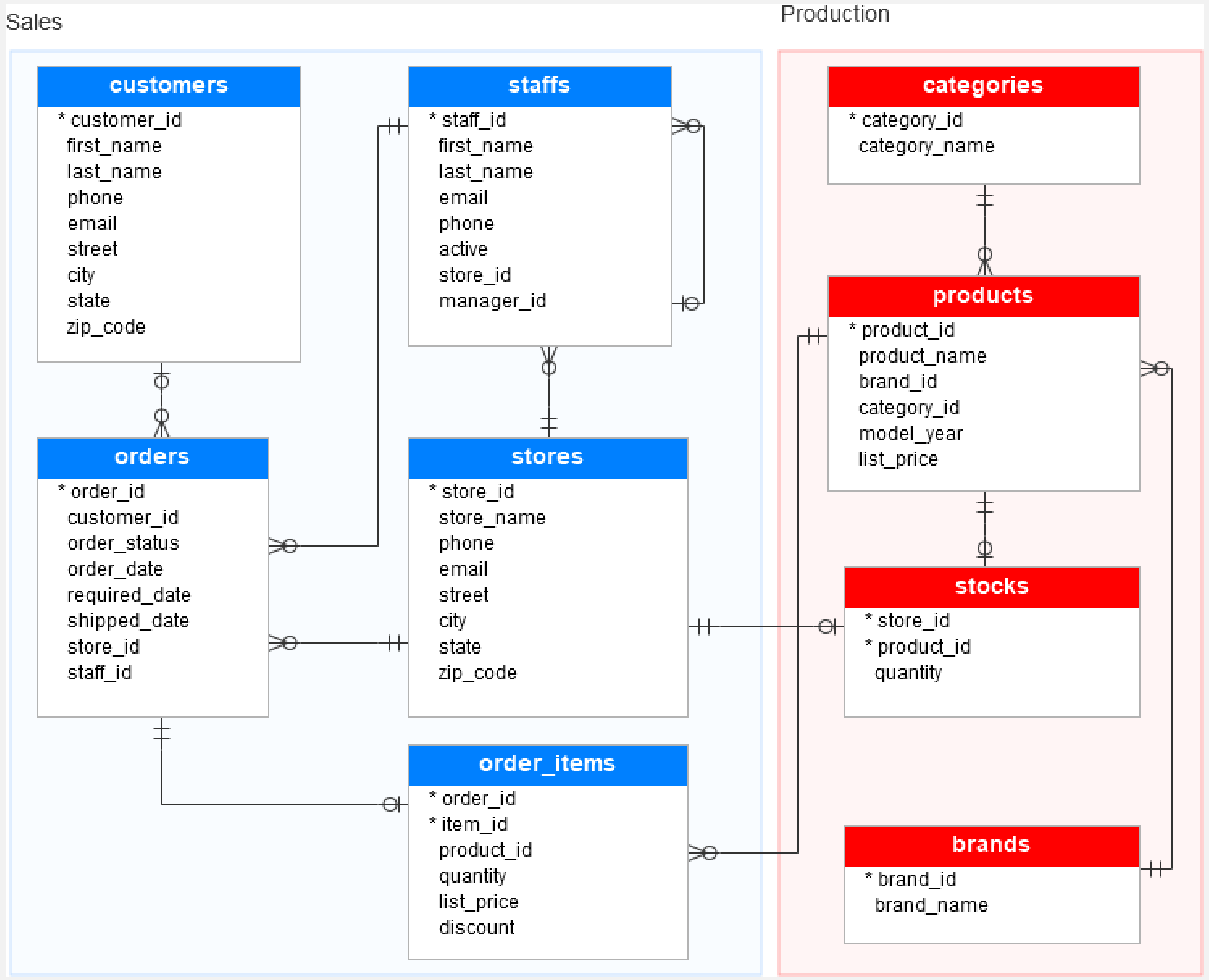


```
CREATE VIEW Name AS
Select column1, Column2...Column N From
tables
Where conditions;
```

SQL Server Date Functions


SQL Server DATE & TIME Functions
Return Date & Time Parts
DATEPART (datepart , date)
DATENAME (datepart , date)
DAY (date)
MONTH (date)
YEAR (date)
Return System Date & Time Values
SYSDATETIME
SYSDATETIMEOFFSET
SYSUTCDATETIME
CURRENT_TIMESTAMP
GETDATE
GETUTCDATE
Return Date & Time Difference Values
DATEDIFF (datepart , startdate , enddate)
DATEDIFF_BIG (datepart , startdate , enddate)
Construct Date & Time
DATEFROMPARTS (year, month, day)
DATETIME2FROMPARTS (year, month, day, hour, minute, seconds,fractions, precision)
DATETIMEOFFSETFROMPARTS
TIMEFROMPARTS
Validate Date & Time Values
ISDATE (expression)
Modify Date & Time Values
DATEADD (datepart , number , date)
EOMONTH (start_date [, month_to_add])
SWITCHOFFSET (DATETIMEOFFSET , time_zone)
TODATETIMEOFFSET (expression , time_zone)

Sample Dataset Used for Examples



Archer

THANK YOU !!!