The Comprehensive SQL Course

Malvik Vaghadia

PathfinderAnalytics

Contents

Relational Databases

What is a Database?

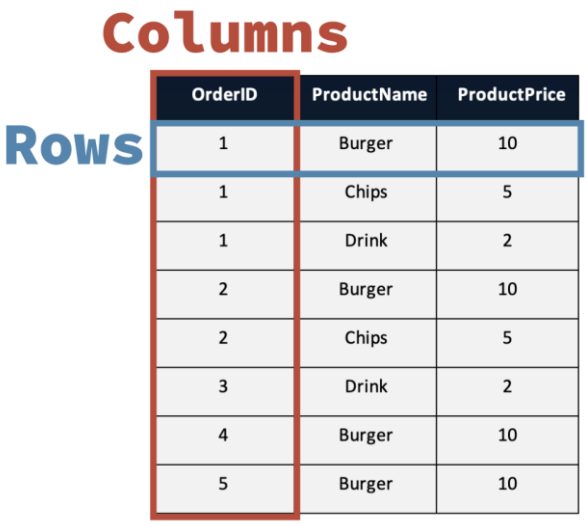
A database is a structured collection of digital information that can be easily accessed, managed, and organized for various purposes, such as storing, retrieving, and manipulating data.

What is a Relational Database?

Relational databases are a specific type of database that organizes and stores data in tables with rows and columns, where relationships between different tables can be established to facilitate efficient data retrieval and management.

What is a Table?

A table is a structured arrangement of data in rows and columns, commonly used to organize and display information.



OrderID	ProductName	ProductPrice
1	Burger	10
1	Chips	5
1	Drink	2
2	Burger	10
2	Chips	5
3	Drink	2
4	Burger	10
5	Burger	10

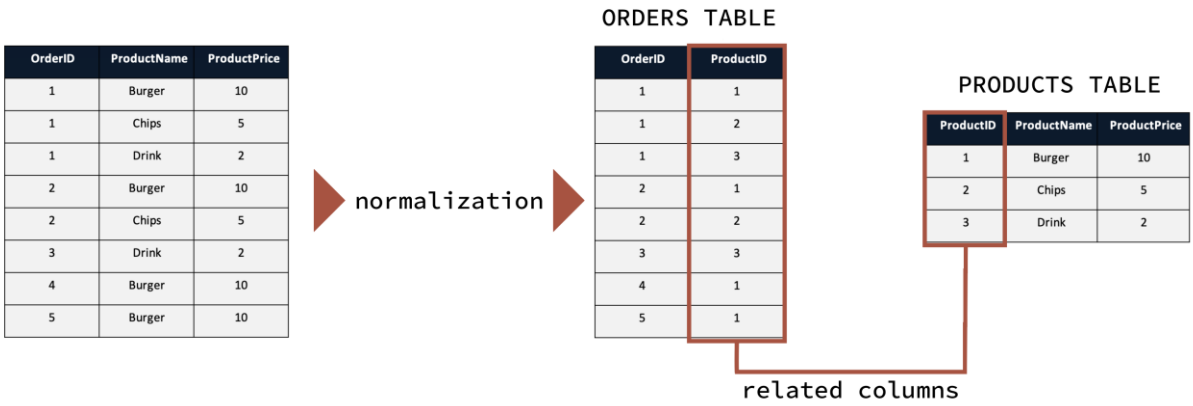
Relational Databases (2)

Benefits of Relational Databases

- Tables adhere to a predefined table structure
  - Data types can restrict the type of values that are inserted into columns
  - Constraints can define rules to determine what kind of values are allowed
- Data Integrity and reliability
  - Data remains accurate, consistent, and adheres to predefined rules, reducing the risk of errors and inconsistencies.
- Databases can store massive amounts of data
  - Relational databases can be implemented on cloud infrastructure and scale up to petabytes of data and beyond

How is data stored in Relational Databases?

Relational Databases store the data in a normalized manner. This means that they organise data into separate tables.



These separate tables are related to each other via a common column. So they have a link to each other which the relational database can leverage when performing queries against the tables.

Relational Databases (3)

Relational Database Management Systems (RDBMS)

An RDBMS is software used to create, manage, and manipulate relational databases. Common RDBMSs include MySQL, PostgreSQL, Oracle and Microsoft SQL Server.

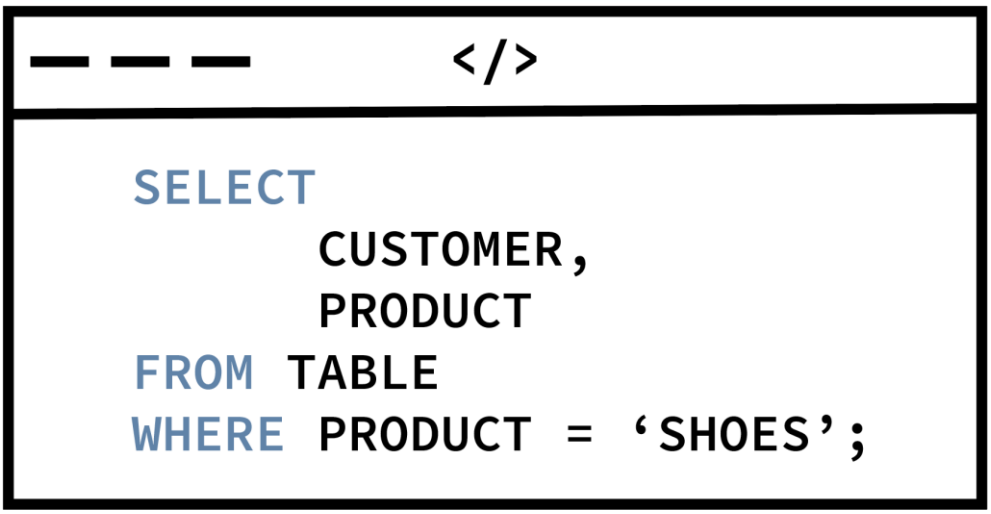
What is MySQL?

- MySQL is an open-source relational database management system
- It’s an open-source software, which means it is freely available for anyone to use, modify, and distribute
- MySQL is available in several editions, including a free, open-source community edition

What is SQL?

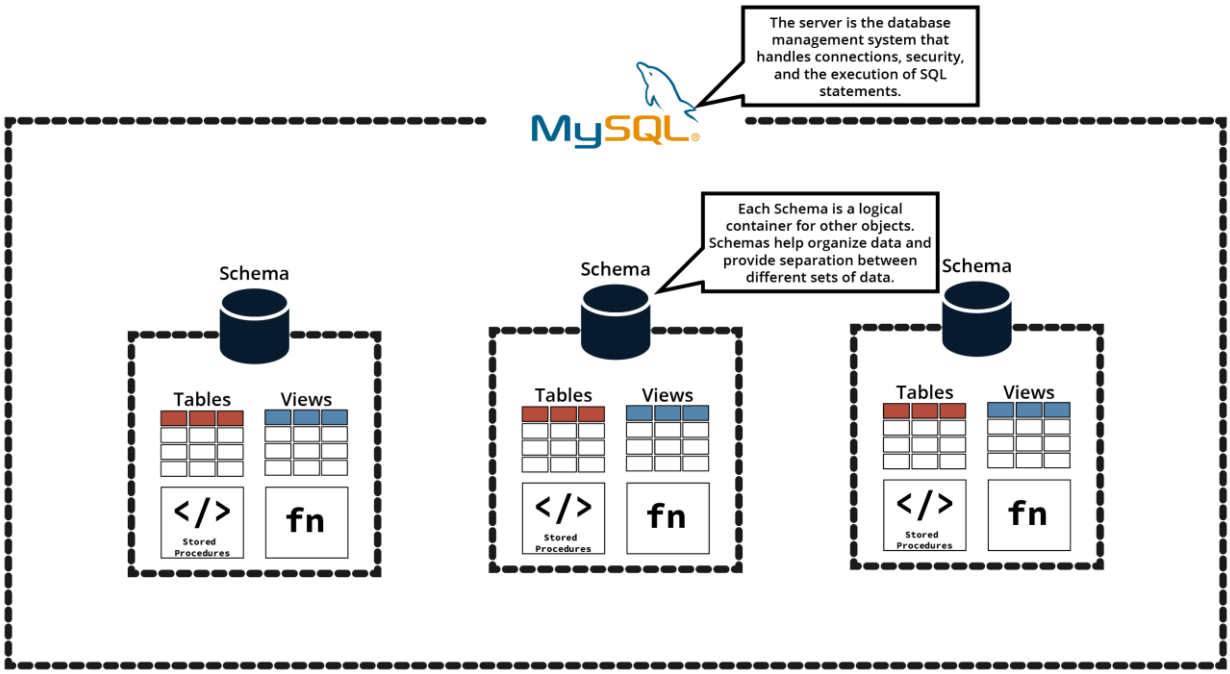
- SQL stands for “Structured Query Language”
- You can pronounce it as “S.Q.L” or “Sequel”
- SQL is the language used to communicate with relational databases
- SQL is a declarative language. You specify the ‘what’ rather than the ‘how’
- This is in contrast with imperative languages like C++, Java and Python, where you specify the ‘how’

Show me all the customers who bought shoes



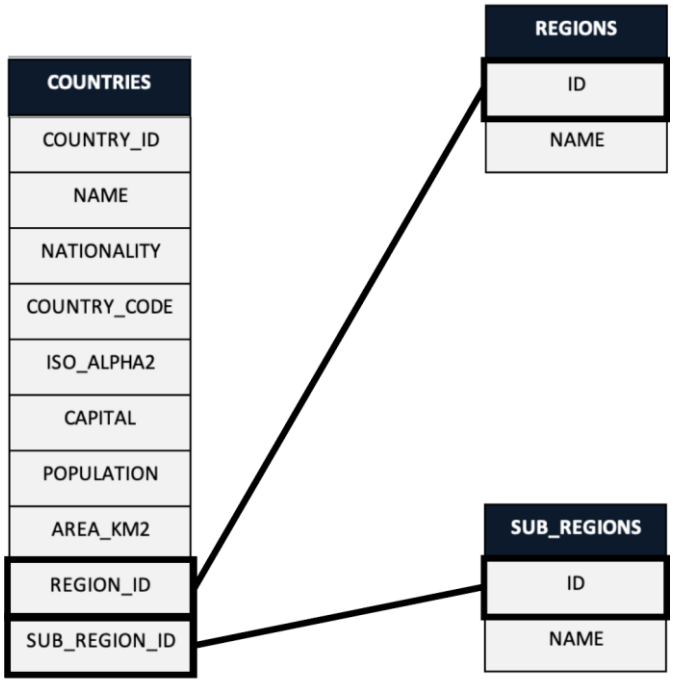
- ANSI stands for American National Standards Institute. It is a set of SQL standards and guidelines for database management systems (DBMS)
- The goal of ANSI SQL is to provide a standardised way to interact with relational databases, making it easier for users to write SQL queries that can work across different database platforms
- Most Relational Database Management Systems will be generally ANSI compliant, but will still have their own slight variations from the ANSI SQL Standards

Objects in MySQL



- At the highest level is the MySQL server itself
- We then have Schemas. Each Schema is a logical container for other objects
- A relational database can have multiple schemas
- Each schema contains objects such as Tables, Views, Stored Procedures and Functions

Countries Data Model

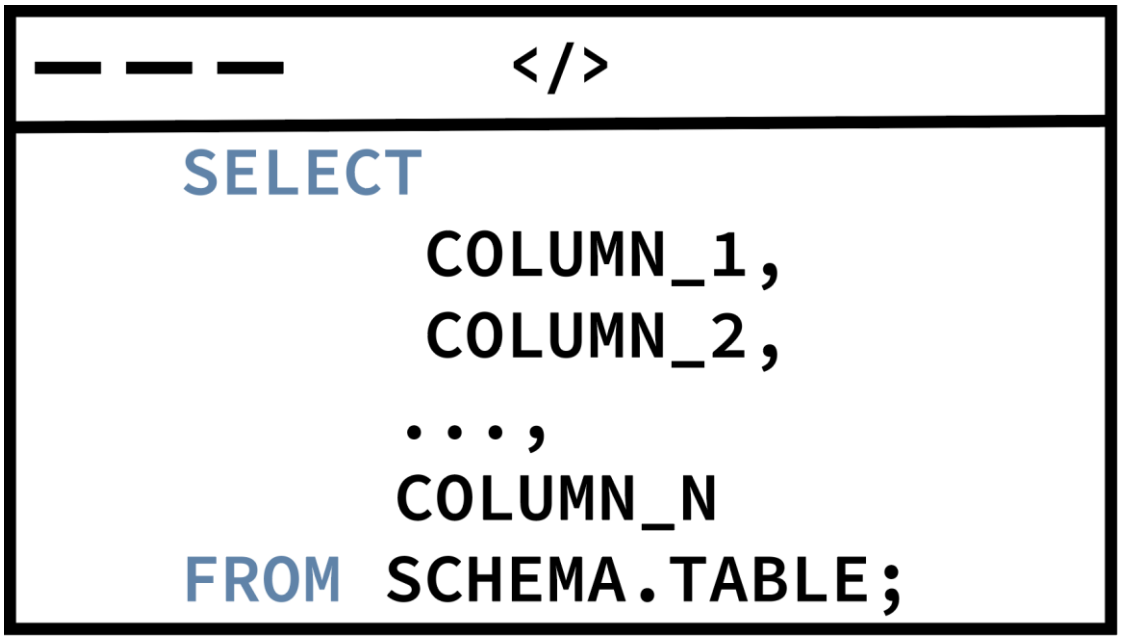


- The “COUNTRIES” table is a Fact Table
- The “REGIONS” and “SUBREGIONS” tables are Dimension Tables
- A Fact Table contains measurements and metrics
- A Dimension Table contains only descriptive attribute

Data Query Language

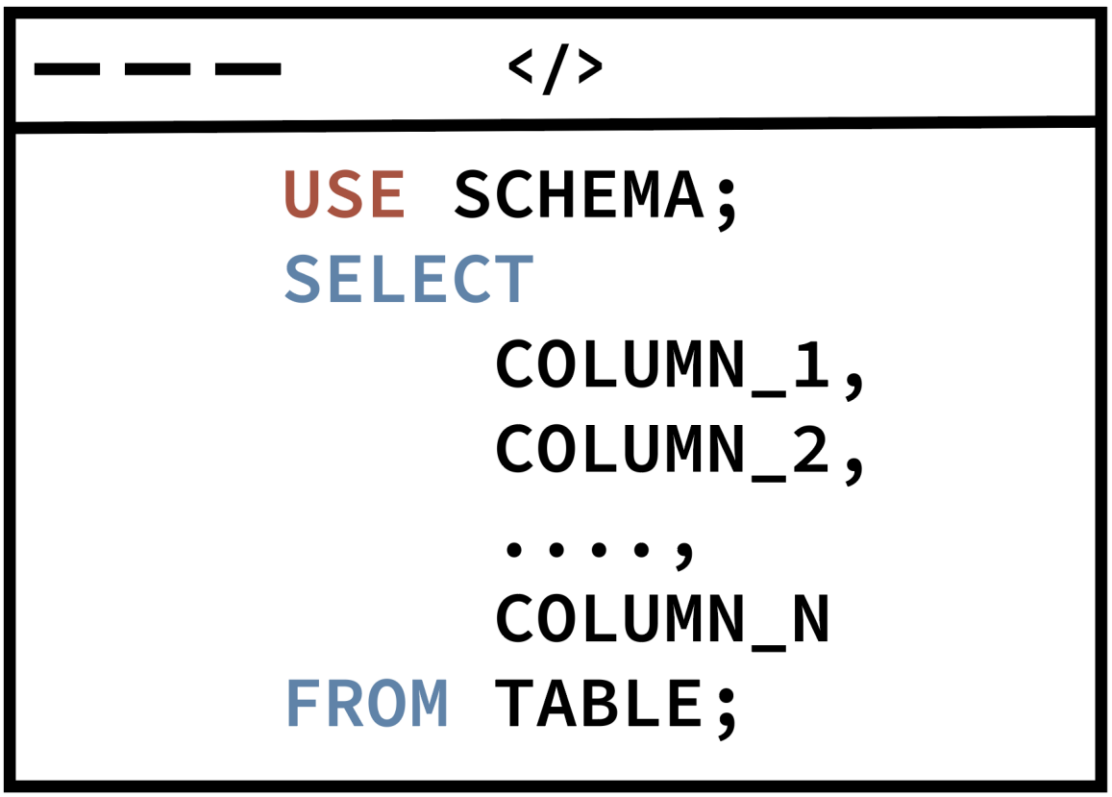
PathfinderAnalytics

The SELECT Statement



- You start with **SELECT** followed by the column names you'd like to return; each column name is separated by a comma. You then type the **FROM** keyword followed by the table reference
- You use dot notation to qualify the table name with the schema to provide context to the SQL engine; this way the SQL engine knows which schema to look in
- Semi-colons (;) are used to terminate an individual statement
- **FROM** specifies the table, **SELECT** specifies the columns
  - The **FROM** operation is executed before the **SELECT** operation

USE SCHEMA

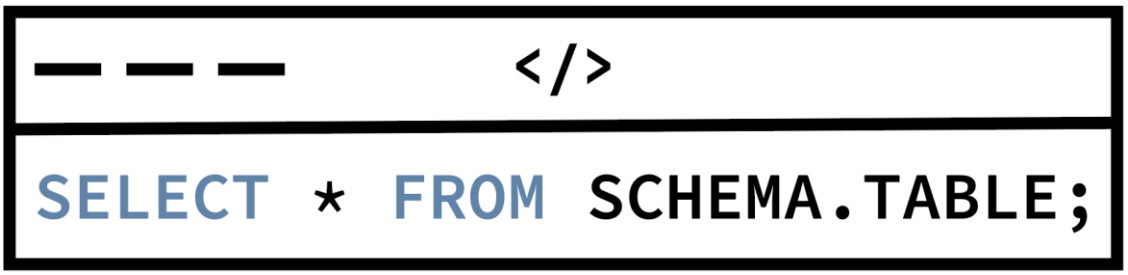


- You can use the "**USE SCHEMA**" command to switch context between schemas, simply type **USE** followed by the schema name
- … you can then avoid using dot notation to qualify the table name with the schema reference

Statements vs Queries

- **“Queries” are a subset of “Statements”**
- A 'statement' refers to any type of code in SQL that can manipulate data, delete data, create tables, and more
- On the other hand, a 'query' is a specific type of SQL statement, namely the '**SELECT**' statement
- Queries are used to retrieve data from a database
- So, while a “Query” is a type of “statement”, a “statement” can encompass a broader range of actions in SQL

SELECT * (STAR)



- Using * with the **SELECT** Statement returns all columns from the table
- This can save the manual effort of having to type out each column
- Using **SELECT Star** is not efficient if you only need to return a few columns. This is because you will retrieve more data than necessary, which can slow down the query

Aliasing Column Names

```
SELECT
      COLUMN_1 AS ALIAS_1,
      COLUMN_2 AS 'ALIAS 2',
      COLUMN_3 AS "ALIAS 3",
      COLUMN_4 ALIAS_4
      ....,
      COLUMN_N
FROM SCHEMA.TABLE;
```

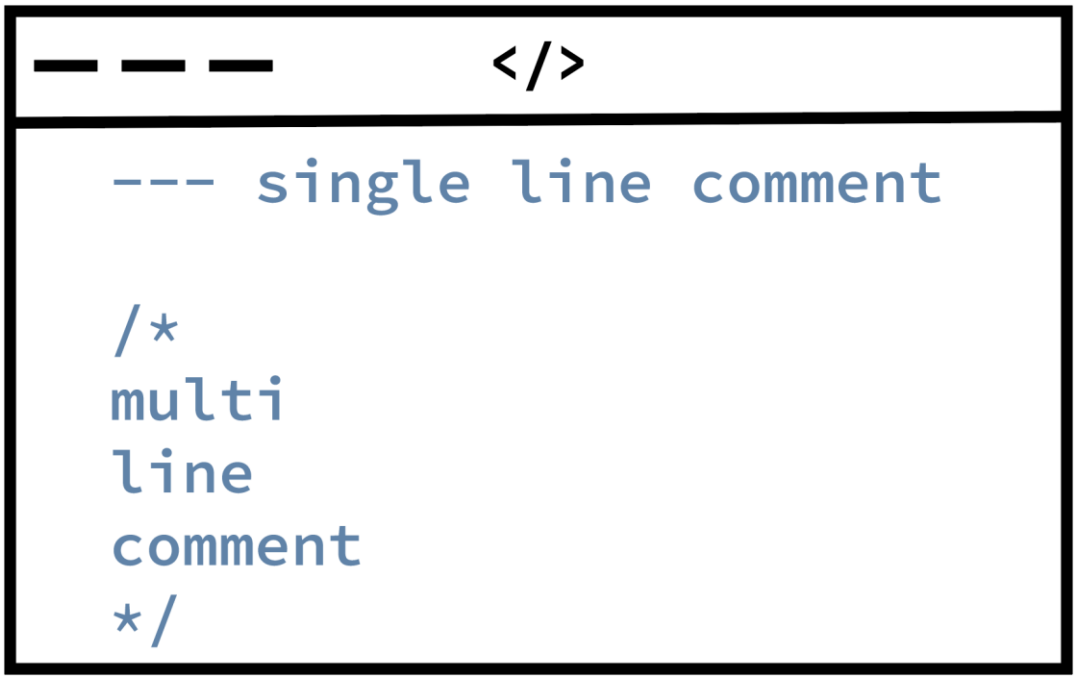
- To alias a column name you simply add the "**AS**" keyword followed by the aliased column name
- Aliasing a column does not change the underlying data
- Specifying the "**AS**" keyword is optional; provided you have a whitespace after the column reference and before the aliased name
- But for good practice and readability it is always recommended to leave the "**AS**" keyword
- You can provide an alias with whitespace characters by using single or double quotes

Aliasing Table Names

```
SELECT
      T1.COLUMN_1,
      T1.COLUMN_2,
      ....,
      T1.COLUMN_N
FROM SCHEMA.TABLE AS T1;
```

- To alias a table name you simply add the "**AS**" keyword followed by the aliased table name
- Aliasing a table does not change the underlying data
- Aliasing a table is useful when working with multiple tables in a single query, this is so you can qualify the column references with the aliased table to make your query syntactically easier
- You should consider the order of execution of SQL clauses

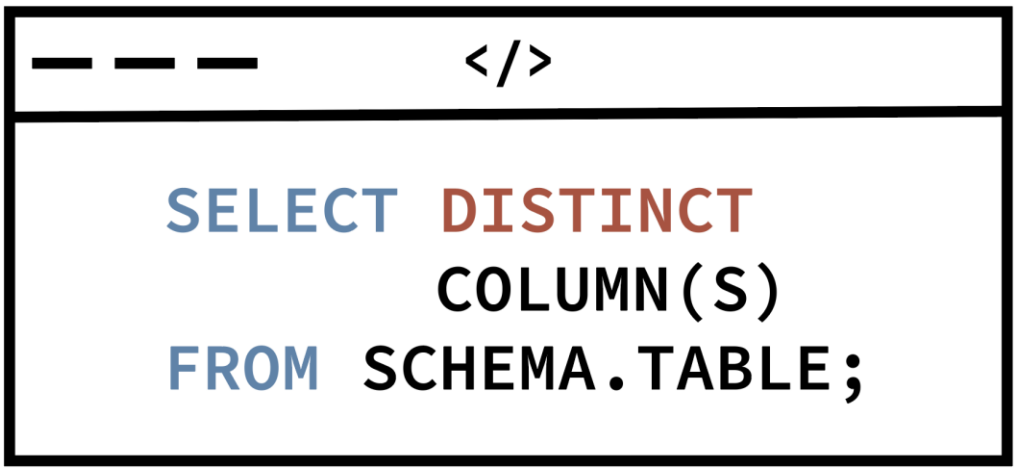
Adding Comments to SQL Code



There are several reasons why code should contain comments.

- They are a way of adding documentation to your code. Comments in SQL code explain the purpose of queries, tables, and columns, making it easier for developers to understand what is going on
- Comments can help identify and resolve issues by providing context, especially in complex SQL queries or when multiple team members work on the same database.
- Well-placed comments enhance the readability of code. They can clarify complex algorithms, logic, or business rules, making the code easier to follow and maintain.
- In a team environment, comments help facilitate collaboration. Team members can quickly grasp your intentions and contribute effectively to the codebase.
- Over time, code undergoes updates, enhancements, and bug fixes. Comments can act as a reference point for making changes without unintentionally altering the code's original purpose. A brief comment can remind you or others why a particular approach was chosen.
- When new developers join a project, comments can be invaluable for getting them up to speed quickly. A brief comment can provide context and reduce the learning curve.

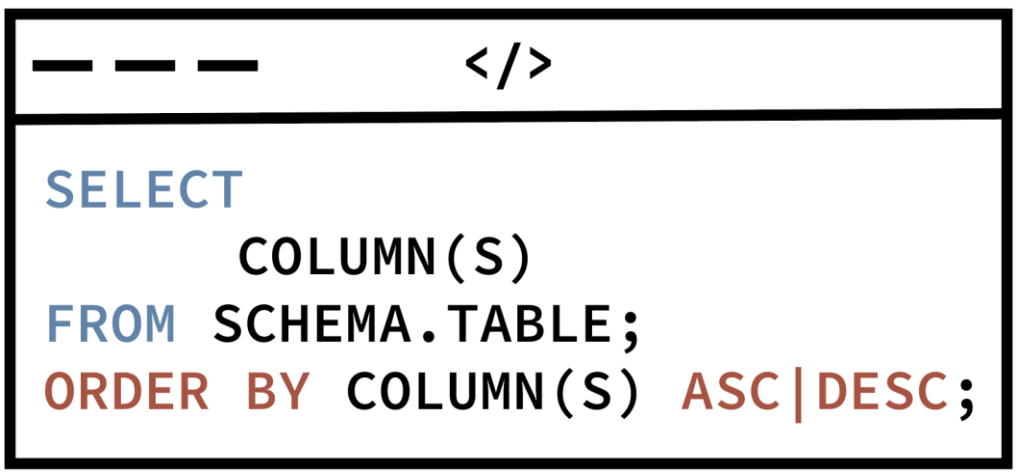
Selecting Distinct Rows



```
SELECT DISTINCT
       COLUMN(S)
FROM SCHEMA.TABLE;
```

- The **SELECT DISTINCT** statement is used to retrieve unique values from a specified column or set of columns in a table. It ensures that the result set contains only unique rows and eliminates duplicate rows.

Ordering Rows



```
SELECT
       COLUMN(S)
FROM SCHEMA.TABLE;
ORDER BY COLUMN(S) ASC|DESC;
```

- You specify the column, or columns that you want to order by after the **ORDER BY** clause. After each column you can specify ASC for ascending or DESC for descending order.
- By default the ordering is ASC if nothing is specified.

Limiting Rows

```
SELECT
       COLUMN(S)
FROM SCHEMA.TABLE;
ORDER BY COLUMN(S) ASC|DESC
LIMIT O,N;
```

- The **LIMIT** clause goes after the **ORDER BY** clause
- O is the offset, if this is omitted then by default the offset is 0
- N is the number of rows

Data Types

PathfinderAnalytics

Numerical Data Types

- An **exact numeric data type**, also known as a fixed-point numeric data type, is a category of data types used to store numbers with a fixed number of decimal places
- In this category we have **INTEGERS**, **DECIMALS** and **NUMERIC**

		Unsigned (negative and positive values)		Signed (positive values only)	
Type	Storage (Bytes)	Minimum Value	Maximum Value	Minimum Value	Maximum Value
TINYINT	1	-128	127	0	255
SMALLINT	2	-32,768	32,767	0	65,535
MEDIUMINT	3	-8,388,608	8,388,607	0	16,777,215
INT	4	-2,147,483,648	2,147,483,647	0	4,294,967,295
BIGINT	8	-9,223,372,036,854,780,000	9,223,372,036,854,780,000	0	18,446,744,073,709,600,000

DECIMAL(p, s) NUMERIC(p, s)	p = precision (the number of significant digits) s = scale (the scale represents the number of digits after following the decimal point) So for example DECIMAL(5,2) or NUMERIC(5,2) can store any value with 5 digits max and up to 2 decimal places So the range of values it can store are between -999.99 to 999.99

- An **approximate numeric data type**, also known as a floating-point numeric data type, is a category of data types used to store numbers with a fractional part or numbers that can have a wide range of values
- In this category we have **FLOAT** and **DOUBLE**
- These data types are considered "approximate" because they represent numbers with a finite level of precision
- However, they are very efficient in terms of storage and computational speed

FLOAT(p, s)	(p,s) means than values can be stored with up to p digits in total, of which s digits may be after the decimal point For example, a column defined as FLOAT(7,4) is displayed as -999.9999
FLOAT(p)	p is Precision. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column
DOUBLE(p, s)	A normal-size floating point number. The total number of digits is specified in p The number of digits after the decimal point is specified in the s parameter

String Data Types

- **CHAR** is short for character, and is a fixed-length character data type
- Each value stored in a CHAR column will always occupy the same amount of storage, regardless of the actual length of the data
- If you insert a shorter string into a CHAR column, MySQL pads the extra space with spaces to match the specified length
  - For example, if you insert the string 'hello' into a CHAR(10) column, it will be stored as 'hello ' (with 4 spaces at the end)
- **VARCHAR** stands for "variable character" and is a variable-length character data type
- Unlike with CHAR, the actual storage used by a VARCHAR column depends on the length of the data you store in it
  - If you insert the string 'hello' into a VARCHAR(10) column, it will be stored as 'hello' (without extra spaces)

Value	CHAR(4)	VARCHAR(4)
''	' '	''
'ab'	'ab '	'ab'
'abcd'	'abcd'	'abcd'
'abcdefgh'	'abcd'	'abcd'

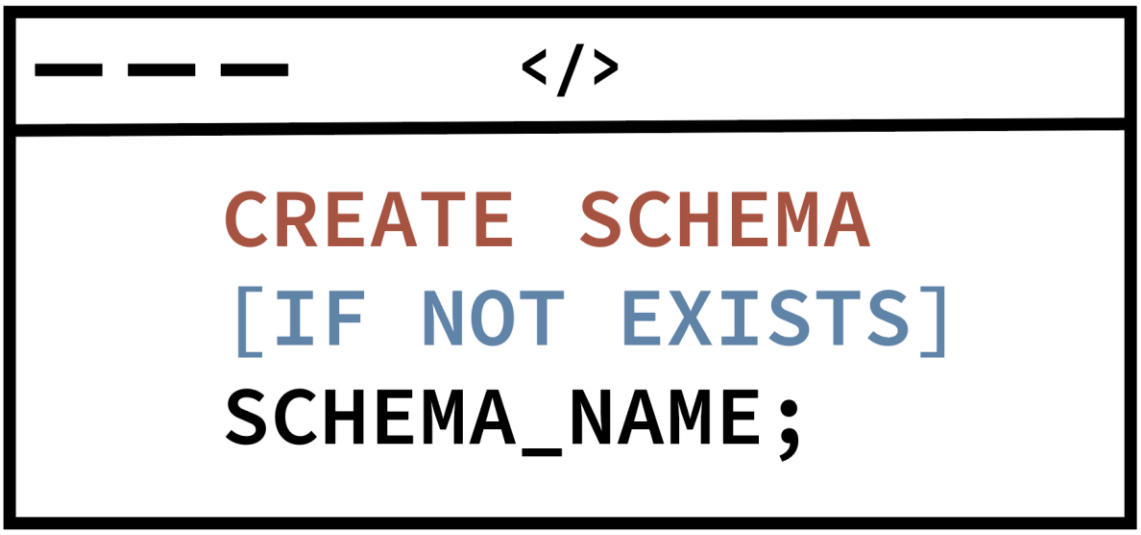
Date and Time Data Types

Data Type	Standard Format	Example
DATE	'YYYY-MM-DD'	'2023-10-02'
TIME	'hh:mm:ss'	'19:39:00'
DATETIME	'YYYY-MM-DD hh:mm:ss'	'2023-10-02 19:39:00'
TIMESTAMP	'YYYY-MM-DD hh:mm:ss'	'2023-10-02 19:39:00'
YEAR	YYYY	'2023'

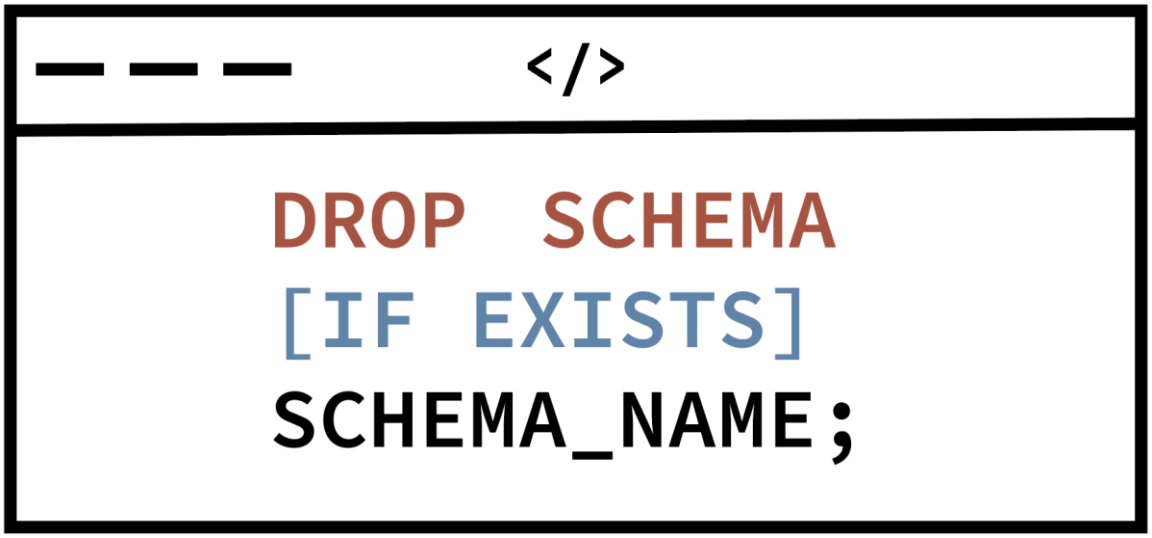
Data Definition and Manipulation Language

PathfinderAnalytics

Creating Schemas



Dropping Schemas



Creating Tables

```
CREATE TABLE [IF NOT EXISTS] SCHEMA.TABLE(
    COLUMN_1 DATATYPE [CONSTRAINTS],
    ...,
    COLUMN_N DATATYPE [CONSTRAINTS]
)
  [OPTIONS]
;
```

- https://dev.mysql.com/doc/refman/8.0/en/create-table.html
- Anything in square brackets is optional

Dropping Tables

```
DROP TABLE [IF EXISTS] SCHEMA.TABLE;
```

- https://dev.mysql.com/doc/refman/8.0/en/alter-table.html
- Anything in square brackets is optional

Inserting Rows

```
INSERT INTO SCHEMA.TABLE
(COLUMN_1, ..., COLUMN_N)
VALUES
(VALUE_1, ..., VALUE_N);
```

- You can insert records into a Database Table by using the **INSERT INTO** statement
- The values you enter must adhere to the data type requirements

```
INSERT INTO SCHEMA.TABLE
VALUES
(VALUE_1, ..., VALUE_N);
```

- If you do not reference the columns in your **INSERT INTO** statement then the values must be specified as per the column order in the table definition

```
INSERT INTO SCHEMA.TABLE
(COLUMN_1, ..., COLUMN_N)
VALUES
(...),
(...),
(...);
```

- You can insert multiple records at a time using the above syntax

Create Table As Statement

```
CREATE TABLE SCHEMA.TABLE
AS
(SELECT STATEMENT);
```

- The **CREATE TABLE AS** (CTAS) statement allows you to create a new table by copying the structure and data from an existing table or the result of a **SELECT** query

Truncate Table

```
TRUNCATE TABLE SCHEMA.TABLE;
```

- The **TRUNCATE TABLE** statement deletes all records from the table but retains the table structure

Alter Table

```
ALTER TABLE (SOME ACTION);

Examples of actions:
ADD COLUMN_NAME DATATYPE
DROP COLUMN COLUMN_NAME
MODIFY COLUMN COLUMN_NAME DATATYPE
RENAME COLUMN OLD_NAME TO NEW_NAME
RENAME TO NEW_TABLE_NAME
...
```

- The **ALTER TABLE** statement is used to modify an existing database table's structure, such as adding, modifying, or dropping columns

Updating Rows

```
UPDATE SCHEMA.TABLE
SET COLUMN_1 = VALUE_1, ...
WHERE [CONDITION];
```

Deleting Rows

```
DELETE FROM SCHEMA.TABLE
WHERE [CONDITION];
```

Constraints in SQL

Constraints are rules and conditions that are applied to tables and their columns to ensure the integrity, accuracy, and consistency of the data stored in a relational database.

```
CREATE TABLE SCHEMA.TABLE(
        COLUMN_1 DATATYPE CONSTRAINT,
        ...,
        COLUMN_N DATATYPE CONSTRAINT
);
```

```
ALTER TABLE SCHEMA.TABLE
MODIFY COLUMN COLUMN_1 DATATYPE CONSTRAINT;
```

- **NOT NULL** prevents NULL values
- **UNIQUE** prevents duplicate values (excluding NULLs)
- **DEFAULT** ensures that a default value is inserted when a value is not explicitly provided for a column
- **CHECK** constraint is used to limit the value range that can be placed in a column.

Dropping Constraints

The **NOT NULL** and **DEFAULT** constraints can be removed by re-defining the column with the constraint using the **ALTER TABLE** statement.

Simply exclude the constraint in the column specification and it will be removed.

```
ALTER TABLE SCHEMA.TABLE
MODIFY COLUMN COLUMN_1 DATATYPE CONSTRAINT;
```

For the **UNIQUE** and **CHECK** constraint you will have to use the syntax below, specifying the unique name given the the constraint.

```
ALTER TABLE SCHEMA.TABLE
DROP CONSTRAINT [CONSTRAINT_NAME];
```

- The **UNIQUE** constraint name is generally the column name with the constraint
- You can identify the **CHECK** constraint name by using the **SHOW CREATE TABLE** statement.

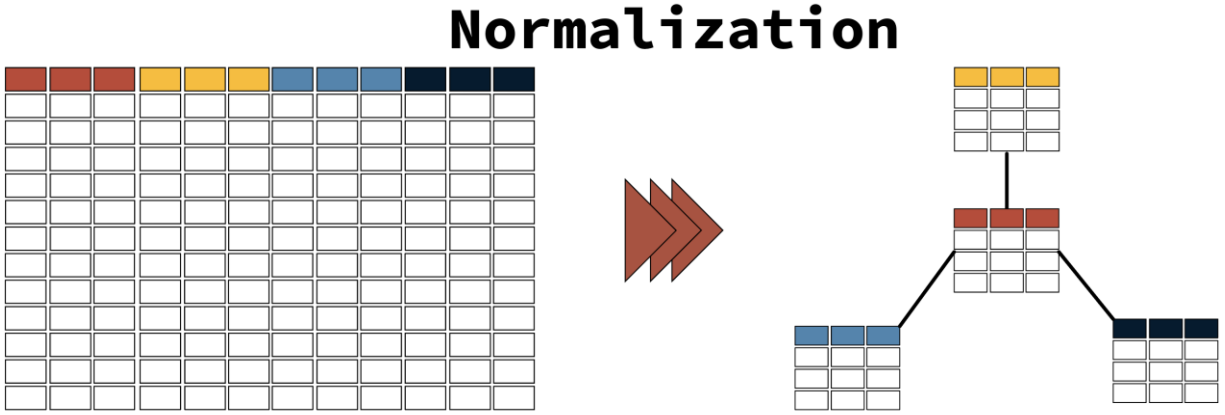
```
SHOW CREATE TABLE SCHEMA.TABLE;
```

Relational Database Design

PathfinderAnalytics

Normalization

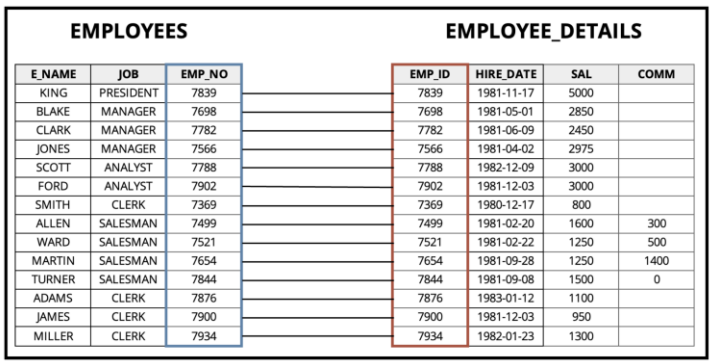
Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. This involves breaking down large tables into smaller ones and using relationships between tables to minimise data duplication



- Data arranged in a normalized manner reduces redundancy and is stored more efficiency
- In some instances it can slow down query performance, but that depends on the type of query
- Denormalization is the opposite of normalization, i.e. combining smaller tables into a larger table

Types of Relationships

- In a **one-to-one relationship**, each record in one table is associated with exactly one record in another table, and vice versa



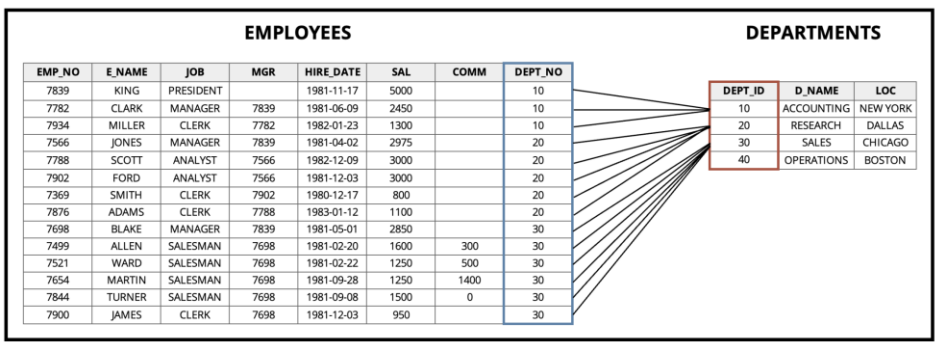
EMPLOYEES

E_NAME	JOB	EMP_NO
KING	PRESIDENT	7839
BLAKE	MANAGER	7698
CLARK	MANAGER	7782
JONES	MANAGER	7566
SCOTT	ANALYST	7788
FORD	ANALYST	7902
SMITH	CLERK	7369
ALLEN	SALESMAN	7499
WARD	SALESMAN	7521
MARTIN	SALESMAN	7654
TURNER	SALESMAN	7844
ADAMS	CLERK	7876
JAMES	CLERK	7900
MILLER	CLERK	7934

EMPLOYEE_DETAILS

EMP_ID	HIRE_DATE	SAL	COMM
7839	1981-11-17	5000	
7698	1981-05-01	2850	
7782	1981-06-09	2450	
7566	1981-04-02	2975	
7788	1982-12-09	3000	
7902	1981-12-03	3000	
7369	1980-12-17	800	
7499	1981-02-20	1600	300
7521	1981-02-22	1250	500
7654	1981-09-28	1250	1400
7844	1981-09-08	1500	0
7876	1983-01-12	1100	
7900	1981-12-03	950	
7934	1982-01-23	1300	

- In a **one-to-many relationship**, each record in one table can be associated with one or more records in another table, but each record in the second table is associated with only one record in the first table



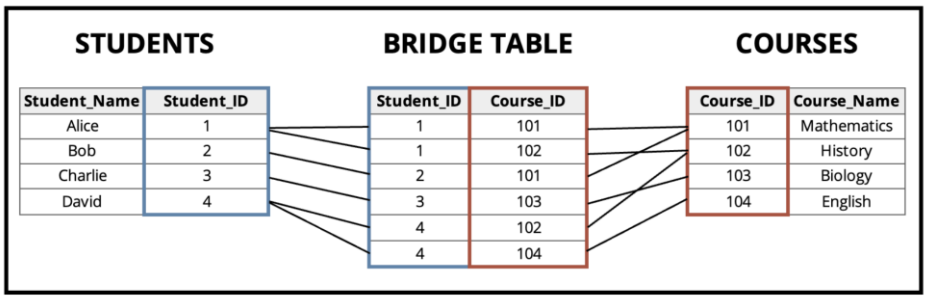
EMPLOYEES

EMP_NO	E_NAME	JOB	MGR	HIRE_DATE	SAL	COMM	DEPT_NO
7839	KING	PRESIDENT		1981-11-17	5000		10
7782	CLARK	MANAGER	7839	1981-06-09	2450		10
7934	MILLER	CLERK	7782	1982-01-23	1300		10
7566	JONES	MANAGER	7839	1981-04-02	2975		20
7788	SCOTT	ANALYST	7566	1982-12-09	3000		20
7902	FORD	ANALYST	7566	1981-12-03	3000		20
7369	SMITH	CLERK	7902	1980-12-17	800		20
7876	ADAMS	CLERK	7788	1983-01-12	1100		20
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30
7900	JAMES	CLERK	7698	1981-12-03	950		30

DEPARTMENTS

DEPT_ID	D_NAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- In a **many-to-many relationship**, many records in one table can be associated with many records in another table, and vice versa. Many-to-many relationships are often implemented using junction tables (also known as bridge tables or linking tables)



STUDENTS

Student_Name	Student_ID
Alice	1
Bob	2
Charlie	3
David	4

BRIDGE TABLE

Student_ID	Course_ID
1	101
1	102
2	101
3	103
4	102
4	104

COURSES

Course_ID	Course_Name
101	Mathematics
102	History
103	Biology
104	English

Primary Keys

Primary keys and foreign keys are essential concepts that are used to establish and maintain relationships between tables and ensure data integrity.

A **primary key** is a column in a table that uniquely identifies each record (row) in that table.

The key characteristics of primary keys are:

- **Uniqueness**: Each value in the primary key column must be unique across all records in the table
- **Non-null**: The values in the primary key column cannot be NULL, ensuring that each record has a unique identifier

```
CREATE TABLE SCHEMA.TABLE(
        COLUMN_1 DATATYPE,
        ...,
        COLUMN_N DATATYPE,
        PRIMARY KEY (COLUMN)
);
```

```
ALTER TABLE SCHEMA.TABLE
ADD PRIMARY KEY (COLUMN);
```

- You cannot drop a PK until its dependencies have been removed

```
ALTER TABLE SCHEMA.TABLE
DROP PRIMARY KEY;
```

Foreign Keys

Primary keys and foreign keys are essential concepts that are used to establish and maintain relationships between tables and ensure data integrity.

A **foreign key** is a column in a table that establishes a link between the data in two tables.

The key characteristics of foreign keys are:

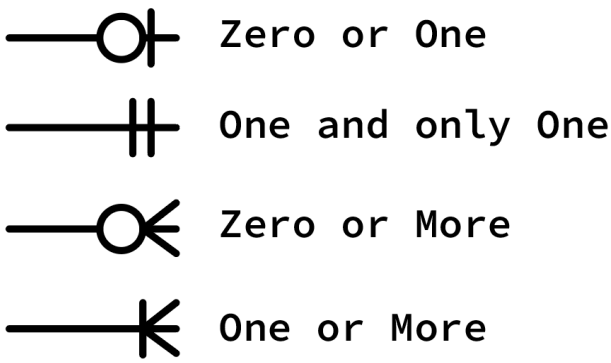
- **Referential Integrity:** A foreign key enforces referential integrity, ensuring that the values in the foreign key column of the table match the values in the primary key column of another table

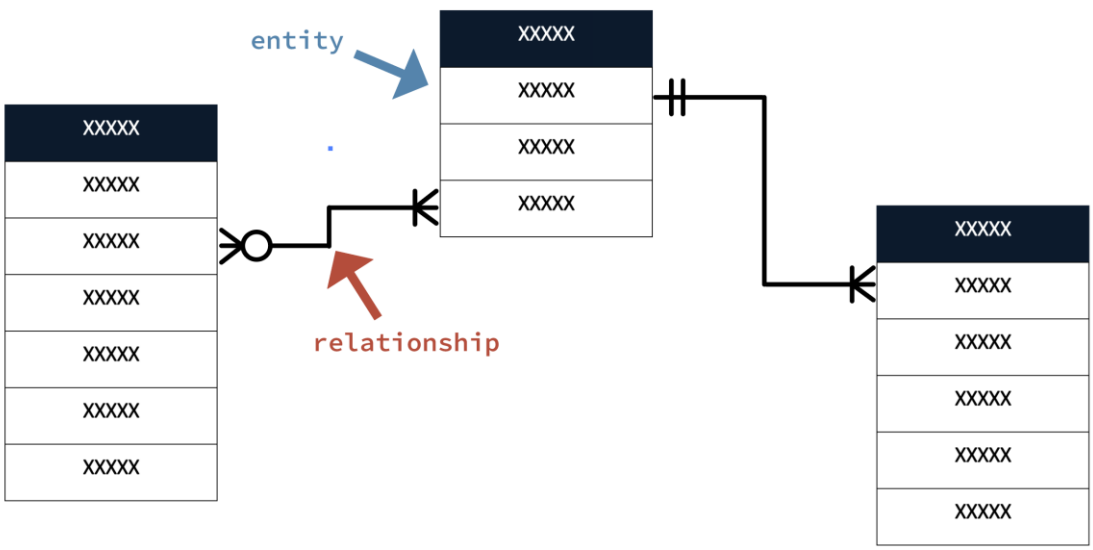
```
CREATE TABLE SCHEMA.TABLE(
COLUMN_1 DATATYPE,
...,
COLUMN_N DATATYPE
FOREIGN KEY (COLUMN) REFERENCES PK_TABLE(PK)
);

ALTER TABLE SCHEMA.TABLE
ADD FOREIGN KEY (COLUMN) REFERENCES PK_TABLE(PK);


ALTER TABLE SCHEMA.TABLE
DROP CONSTRAINT FK_NAME;
```

Entity Relationship Diagram

Entity relationship diagrams (**ERD**) help us understand the connection between various Tables ("entities") that make up a Schema.

The most common type of notation is **Crow's Foot**.





Operators and the Where Clause

The Where Clause

- You can use the **WHERE** clause in **SELECT**, **UPDATE** and **DELETE** statements

```
SELECT * FROM SCHEMA.TABLE
WHERE CONDITION;

UPDATE SCHEMA.TABLE
SET ...
WHERE CONDITION;

DELETE FROM SCHEMA.TABLE
WHERE CONDITION;
```

Arithmetic Operators	
+	Add
-	Subtract
*	Multiply
/	Divide

Comparison Operators	
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical Operators	
AND	ALL all the conditions separated by AND is TRUE
OR	ANY of the conditions separated by OR is TRUE
BETWEEN	Operand is within the range of comparisons
IN	Operand is equal to one of a list of expressions
LIKE	Operand matches a pattern
NOT	Condition(s) is NOT TRUE

Order of Operator Precedence

Internal and Metadata Schemas

PathfinderAnalytics

Internal and Metadata Schemas

sys

- Built-in schema
- Internal performance and monitoring
- Views and procedures providing insights on performance and resource usage

information_schema

Contains a set of tables that store metadata about the database.

- Retrieving information about tables columns, constraints etc
- Querying metadata about user accounts and priviledges
- Examining information about routines
- Character sets and collations

performance_schema

Focuses on performance data and monitors MySQL Server execution.

- Profiling and optimizing SQL queries by examining execution statistics
- Identifying performance bottlenecks
- Monitoring resource usage
- ...

mysql

The MySQL schema is used for managing user accounts, privileges, and other system-related information.

- Managing and configuring MySQL server
- Creating, modifying, removing user accounts
- Priviledges and access control
- Monitoring and managing server logs and errors

Information Schema Table Reference

Functions

Functions

Links

Functions and Operators

String Functions

Numerical Functions

Date and Time Functions

Aggregate Functions

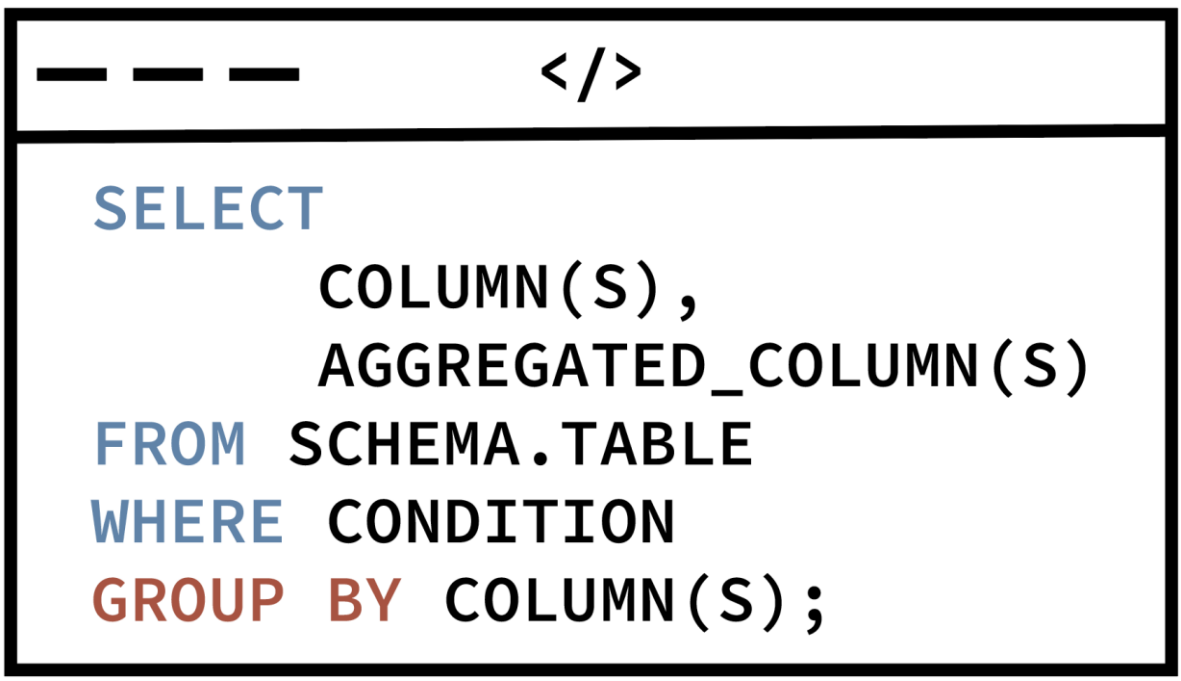
Grouping and Aggregating Data

PathfinderAnalytics

Group By Clause

The **GROUP BY** clause in SQL is used with the **SELECT** statement to group rows that have the same values in specified columns.

It is often used with aggregate functions like **COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**, and others to perform an operation on each group of rows.



```
SELECT
      COLUMN(S),
      AGGREGATED_COLUMN(S)
FROM SCHEMA.TABLE
WHERE CONDITION
GROUP BY COLUMN(S);
```

The non-aggregated columns in the **SELECT** clause must be in the **GROUP BY** clause.

Having Clause

The **WHERE** clause filters records from a result set based on a condition(s).

The **HAVING** clause is very similar, except it filters records after a table has been aggregated.

The **HAVING** clause can only be used in conjunction with the **GROUP BY** clause.

Syntax:

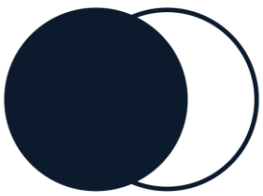
```
SELECT
      COLUMN(S),
      AGGREGATED_COLUMN(S)
FROM SCHEMA.TABLE
WHERE CONDITION -- filters pre-aggregated data
GROUP BY COLUMN(S) -- aggregates data
HAVING CONDITION; -- filters aggregated data
```

Joining Tables and Set Operators

Joining Tables

A **JOIN** operation in SQL is used to combine rows from two or more tables based on a related column between them. It enables the retrieval of data that resides in multiple tables in a single query.

Left Join



Return ALL records from the left table.

Only matched records from the right table.

Unmatched records from the left table will contain NULL values for columns from the right table.

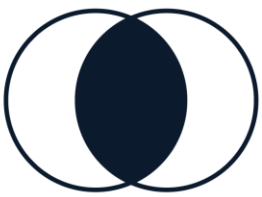
Right Join

Return ALL records from the right table.

Only matched records from the left table.

Unmatched records from the right table will contain NULL values for columns from the left table.

Inner Join



Return only matching records from BOTH tables.

Outer Join

Return ALL records from both tables.

Syntax

```
SELECT
      LEFT_TABLE.COLUMN(S),
      RIGHT_TABLE.COLUMN(S)
FROM LEFT_TABLE
[INNER|LEFT|RIGHT|...] JOIN RIGHT_TABLE
ON LEFT_TABLE.COLUMN = RIGHT_TABLE.COLUMN;
```

JOIN syntax with other clauses

The **JOIN** operation can be used in combination with other SQL clauses.

Syntax

```
SELECT...
FROM...
JOIN...
ON...
WHERE...
GROUP BY...
HAVING...
ORDER BY...
LIMIT...
```

Set Operators

The **UNION** and **UNION ALL** operators combine all results from two query blocks into a single result. **UNION** omits any duplicates, **UNION ALL** does not.

The **INTERSECT** operator combines only those rows where the results of two statements have in common, omitting any duplicates.

The **EXCEPT** operator returns results from the first statement which are not present in the second statement.

Syntax

```
SELECT ...
UNION|UNION ALL|INTERSECT|EXCEPT
SELECT ...;
```

CROSS JOIN returns the Cartesian product of two tables, combining each row of the first table with each row of the second

Order of SQL Operations

PathfinderAnalytics

Order of SQL Execution

FROM

JOIN

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

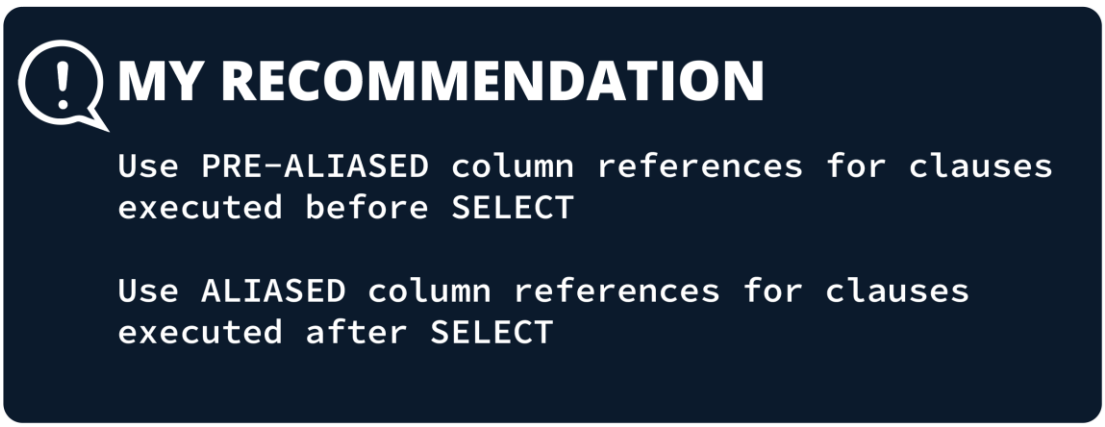
LIMIT

Column Aliasing and SQL Execution Order

The **WHERE** clause will only accept pre-aliased column references.

The **GROUP BY**, **HAVING** and **ORDER BY** clauses can accept both pre-aliased or aliased column references.

This behaviour is specific to MySQL and may not be supported in other SQL databases.

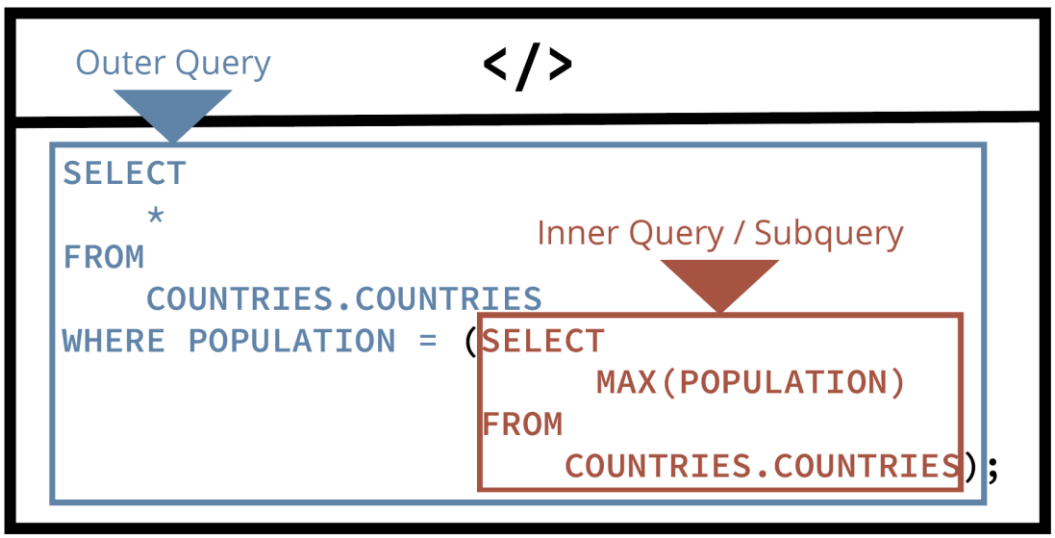


Subqueries and Views

PathfinderAnalytics

Subqueries

- **Subqueries** are queries nested within queries
- They allow you to use the result of one query as a condition or part of another query.
- They can perform operations that usually require multiple queries and structure these queries in a more efficient, streamlined manner.
- Subqueries can be used in various clauses like **SELECT**, **FROM**, **JOIN**, **WHERE**, and **HAVING**.
- They're especially useful for performing complex filtering, aggregating data, or comparing values against a subset of data.
- Subqueries must be enclosed in parentheses and can return single or multiple rows, depending on their placement and purpose.



- ✓ Simplify complex queries
- ✓ Improve readability of code
- ✓ Versatile functionality
- ✗ Can be slower
- ✗ Can be difficult to maintain
- ✗ Can be resource intensive

Views

In SQL **Views** are virtual tables that represent data from one or more tables.

SQL **Views** are constructed from a query.

They be queried like a regular table, but they do not store any physical data; they simply reference underlying tables.

Views offer several benefits such as:

- Simplification of complex code
- Data Abstraction
- Security
- Storage

Syntax:

```
-- create view

CREATE VIEW view_name AS
SELECT ...;

-- update view

CREATE OR REPLACE VIEW view_name AS
SELECT ...;


-- drop view

DROP VIEW view_name;
```

Data Control Language

PathfinderAnalytics

Grant and Revoke

You can create a new user via the Administration Pane on MySQL or via the **CREATE USER** statement.

You can **GRANT** and **REVOKE** privileges to a user for database objects via the Administration Pane on MySQL or via the SQL statements.

Create User Syntax

```
CREATE USER 'username'@'hostname'
IDENTIFIED BY 'password';
```

Grant and Revoke Syntax

```
GRANT PRIVILEGES ON OBJECT TO 'username'@'hostname';

REVOKE PRIVILEGES ON OBJECT FROM 'username'@'hostname';
```

Transaction Control Language

Transaction Control Language

Transaction Control Language (TCL) in SQL is used for managing transactions in a relational database. It allows you to control and maintain the integrity and consistency of your database by defining the boundaries of transactions.

Key TCL statements include:

- **COMMIT**, which confirms and saves the changes made within a transaction to the database
- **ROLLBACK**, which undoes the changes made within a transaction, returning the database to its previous state
- **SAVEPOINT** which allows you to undo the transactions up to a specified point

A transaction can be any data manipulation language operation, such as **INSERT**, **UPDATE** or **DELETE**.

Transactions in relational databases adhere to the **ACID** properties which are:

- Atomicity: All operations in the transaction are treated as a single unit, which either completes fully or not at all
- Consistency: The transaction brings the database from one consistent state to another
- Isolation: Concurrent transactions are isolated from each other, ensuring that their concurrent execution results in a system state that would be obtained if the transactions were executed serially
- Durability: Once a transaction has been committed, its effects are permanent

Commit, Rollback and Savepoint

By default MySQL automatically commits every individual transaction.

To prevent this you can start a transaction with:

BEGIN; or

START TRANSACTION;

Syntax

Commit a transaction:

COMMIT;

Rollback a transaction (you cannot rollback once committed):

ROLLBACK;

Create a savepoint:

SAVEPOINT savepoint_name;

Rollback to a savepoint:

ROLLBACK TO SAVEPOINT savepoint_name;