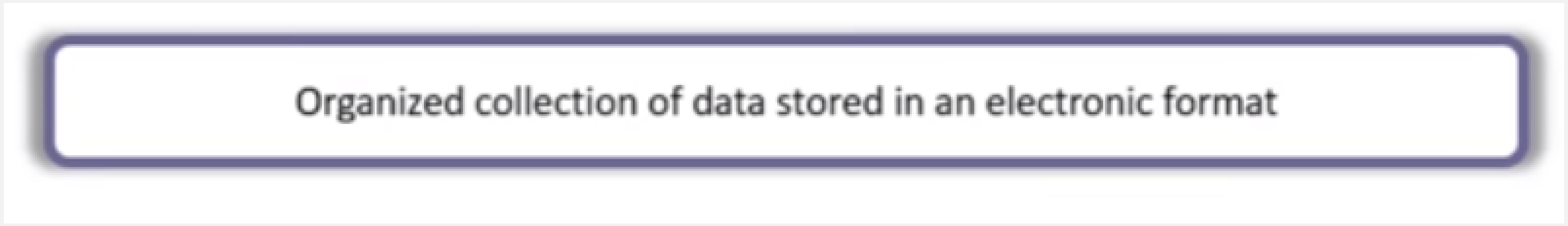
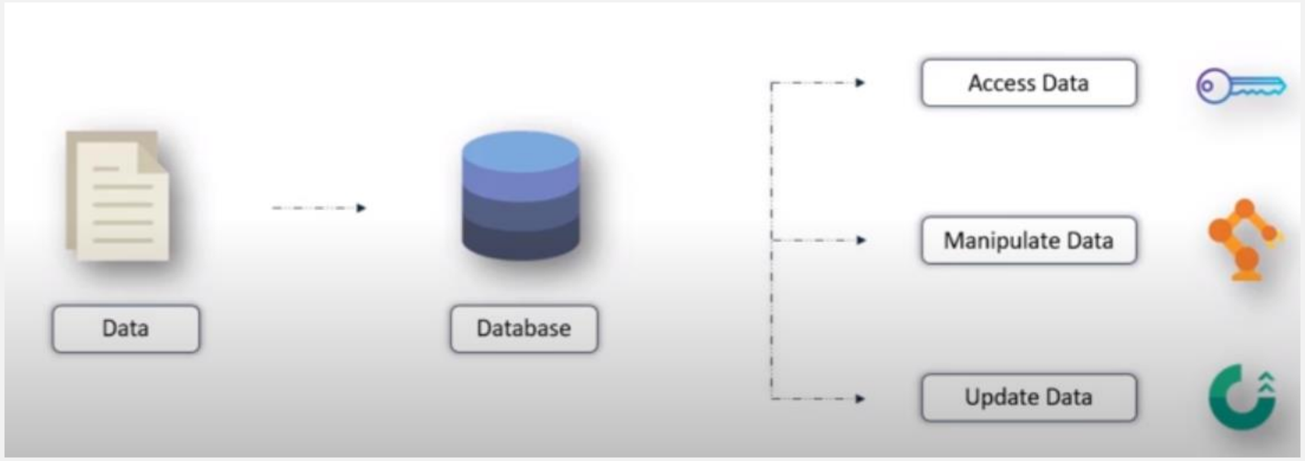
MYSQL Basics

Archer Infotech , PUNE

What is Database ?

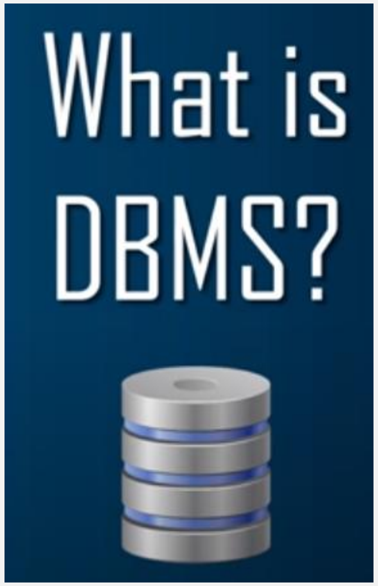


Organized collection of data stored in an electronic format



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

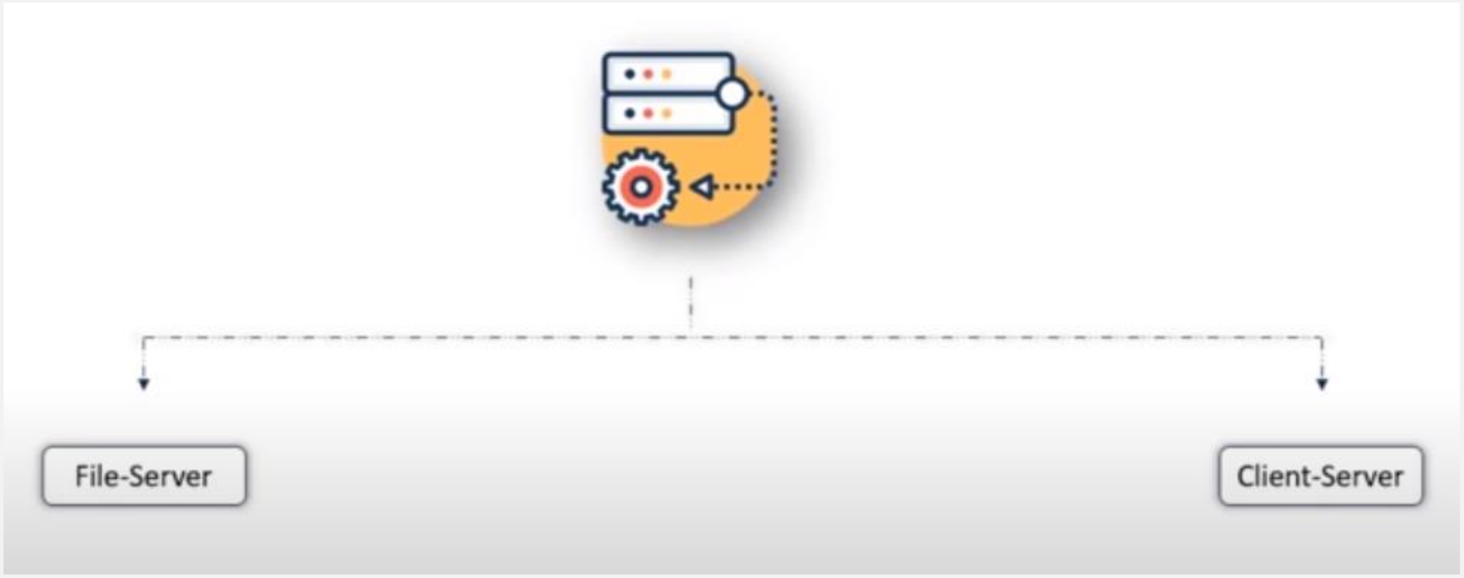
What is DBMS ?



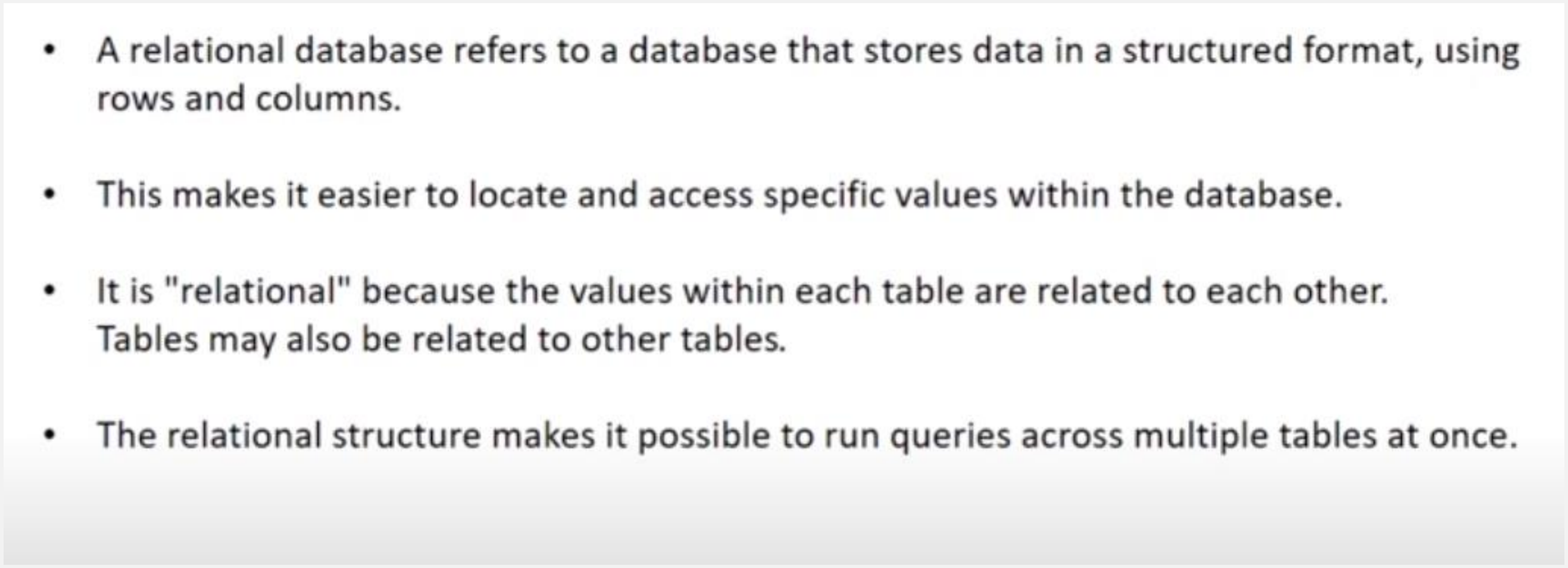
A Database Management System (DBMS) is a software that is used to manage the Database.

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

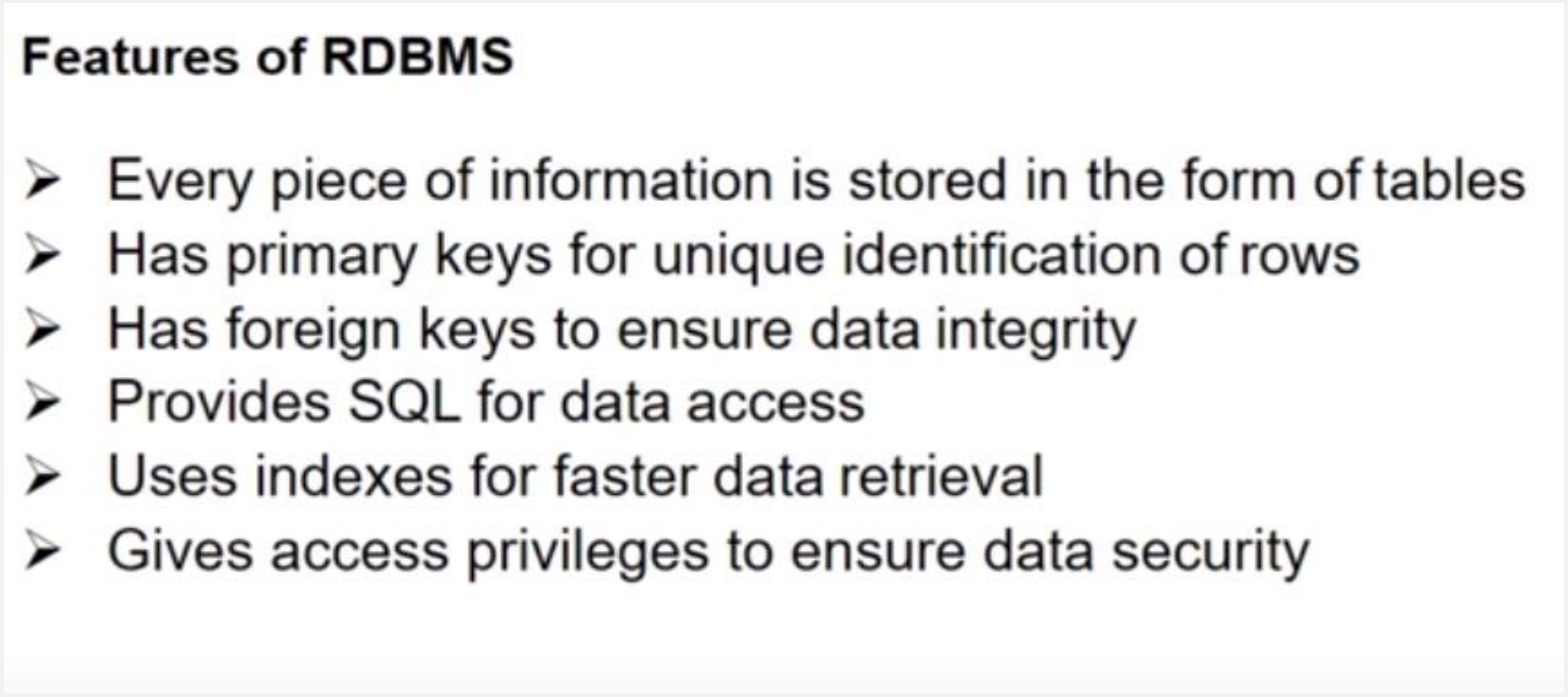


RDBMS



- 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

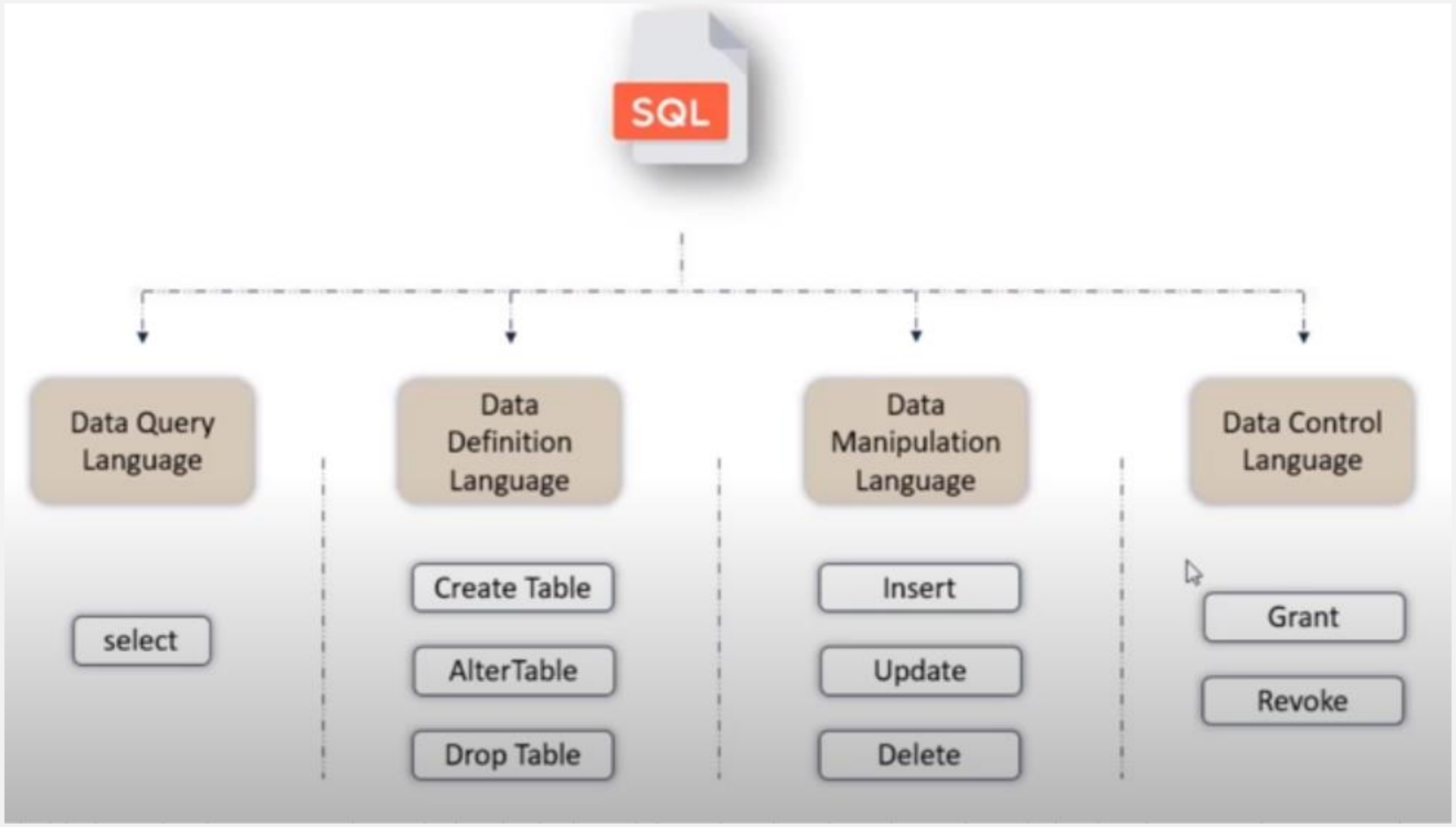
MySQL
ORACLE
Microsoft SQL Server
SYBASE

Introduction to SQL

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

SQL is pronounced as "See-Quel"

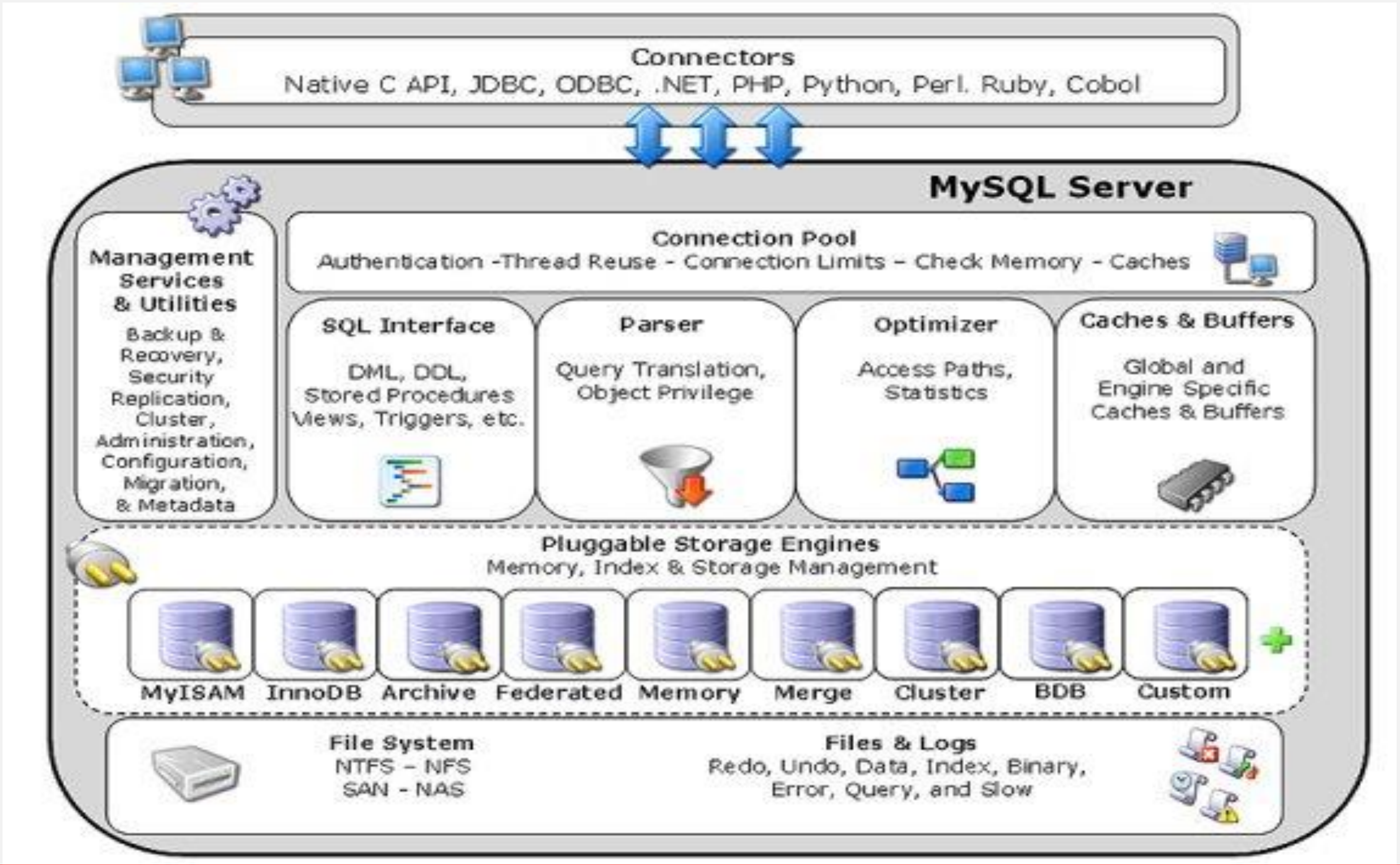
SQL Command Categories



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

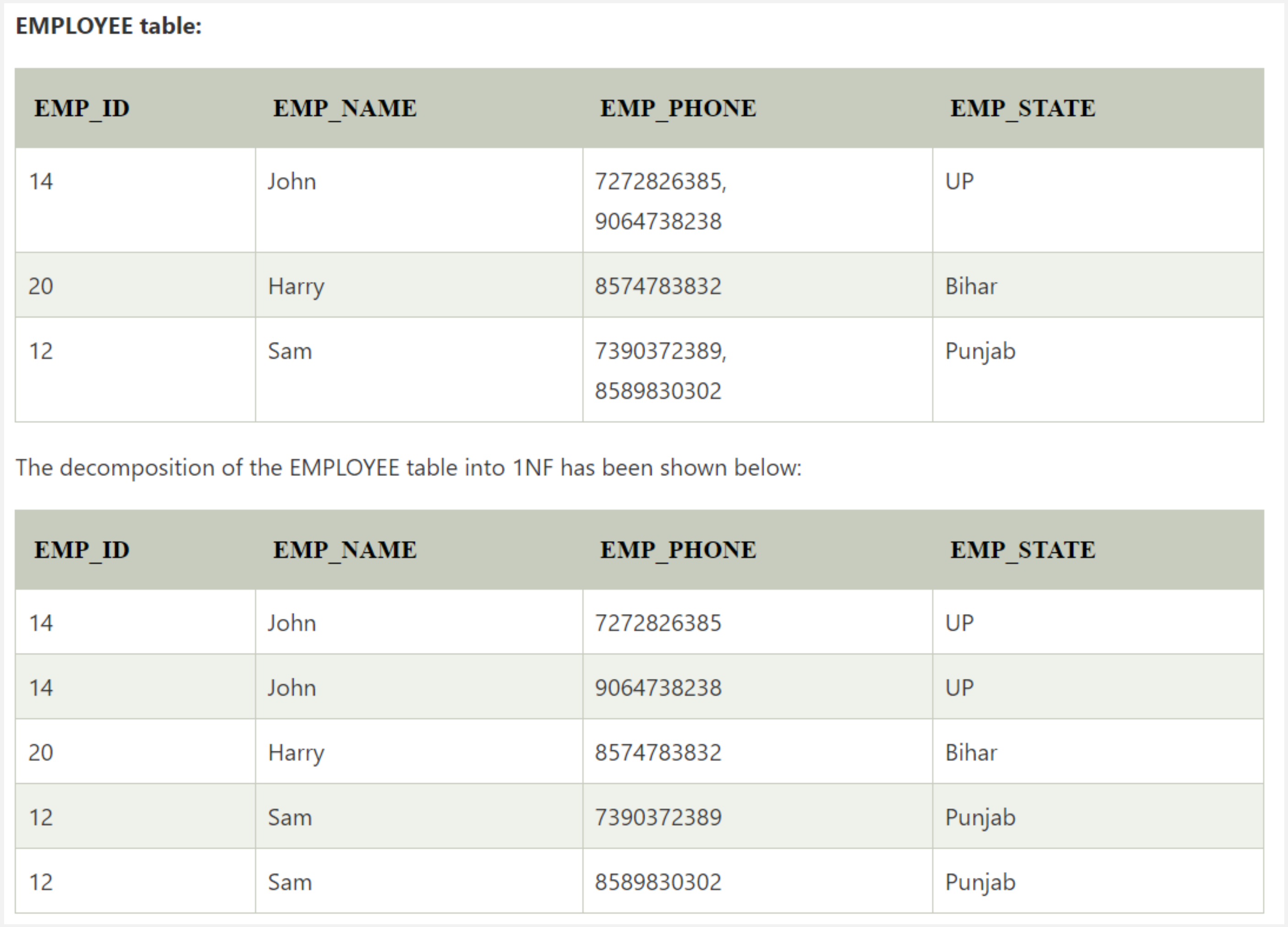
Here is a list of Normal Forms in SQL:

- 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:

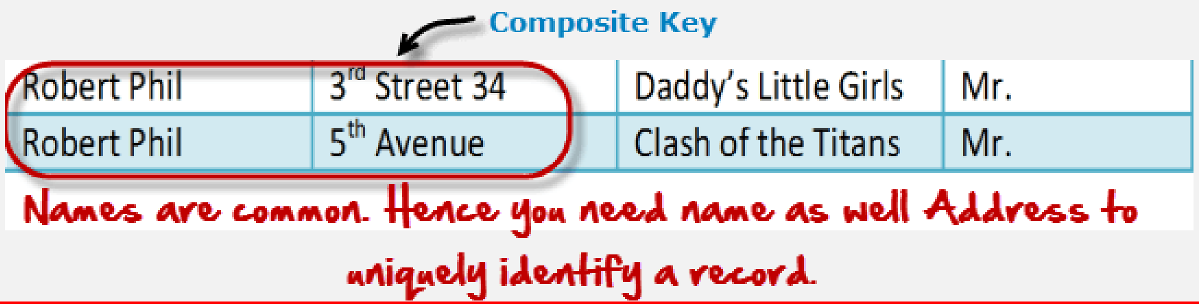
EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

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

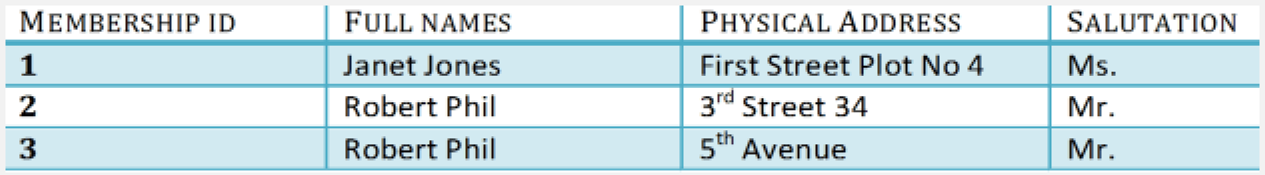
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

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

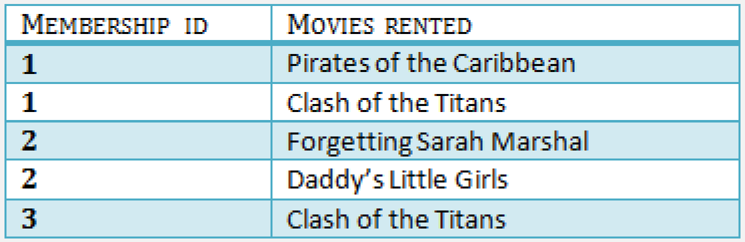
2NF (Second Normal Forms) Rules

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.

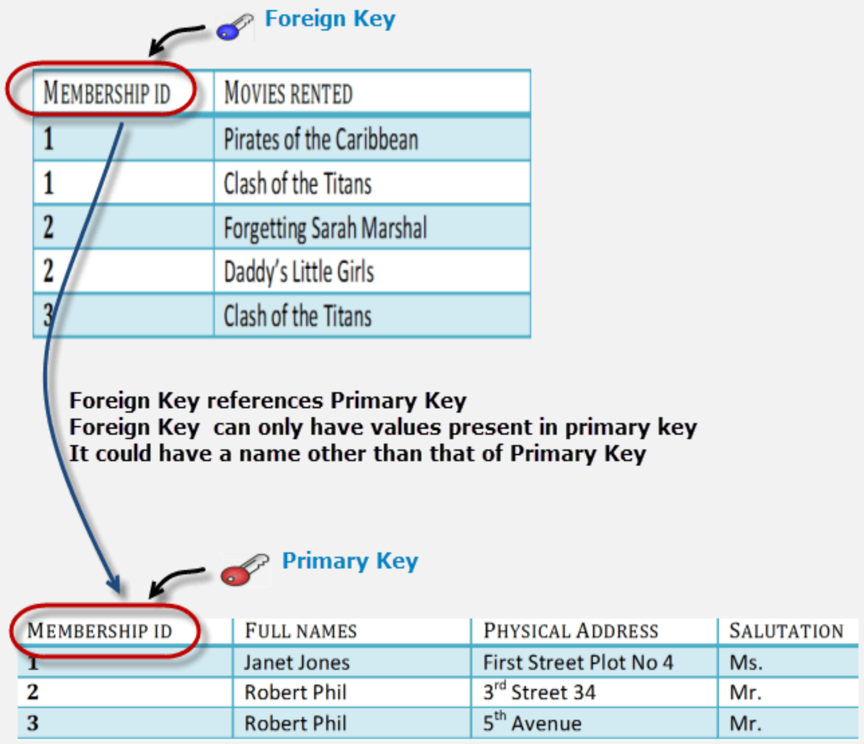


MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

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



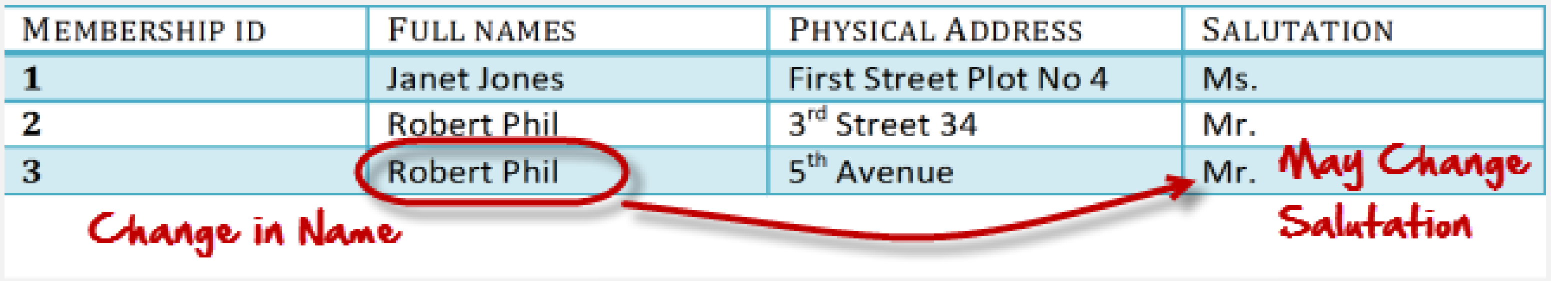
MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

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



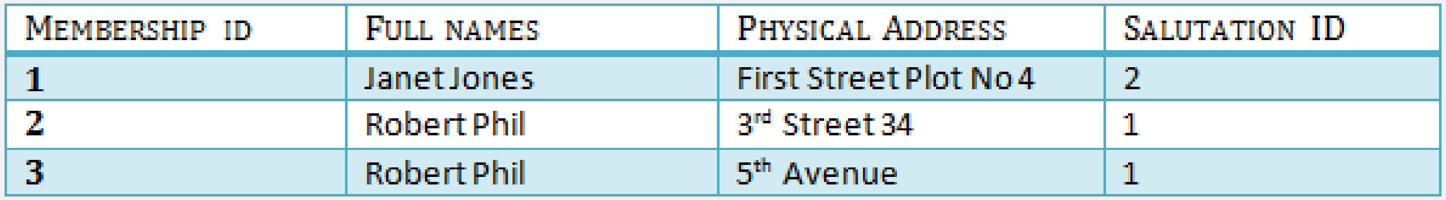
MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Change in Name
May Change Salutation

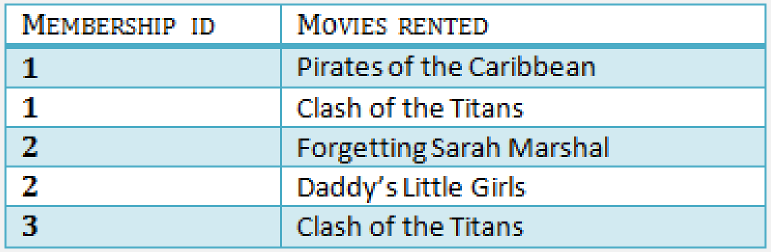
3NF (Third Normal Form) Rules

Rule 1- Be in 2NF

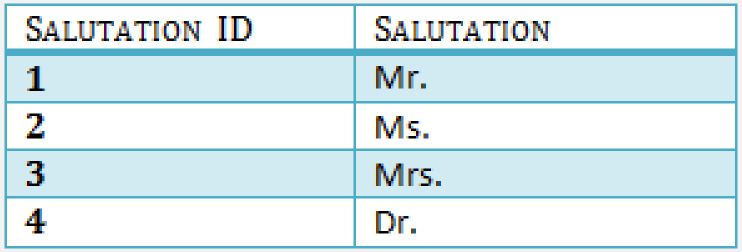
Rule 2- Has no transitive functional dependencies



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).

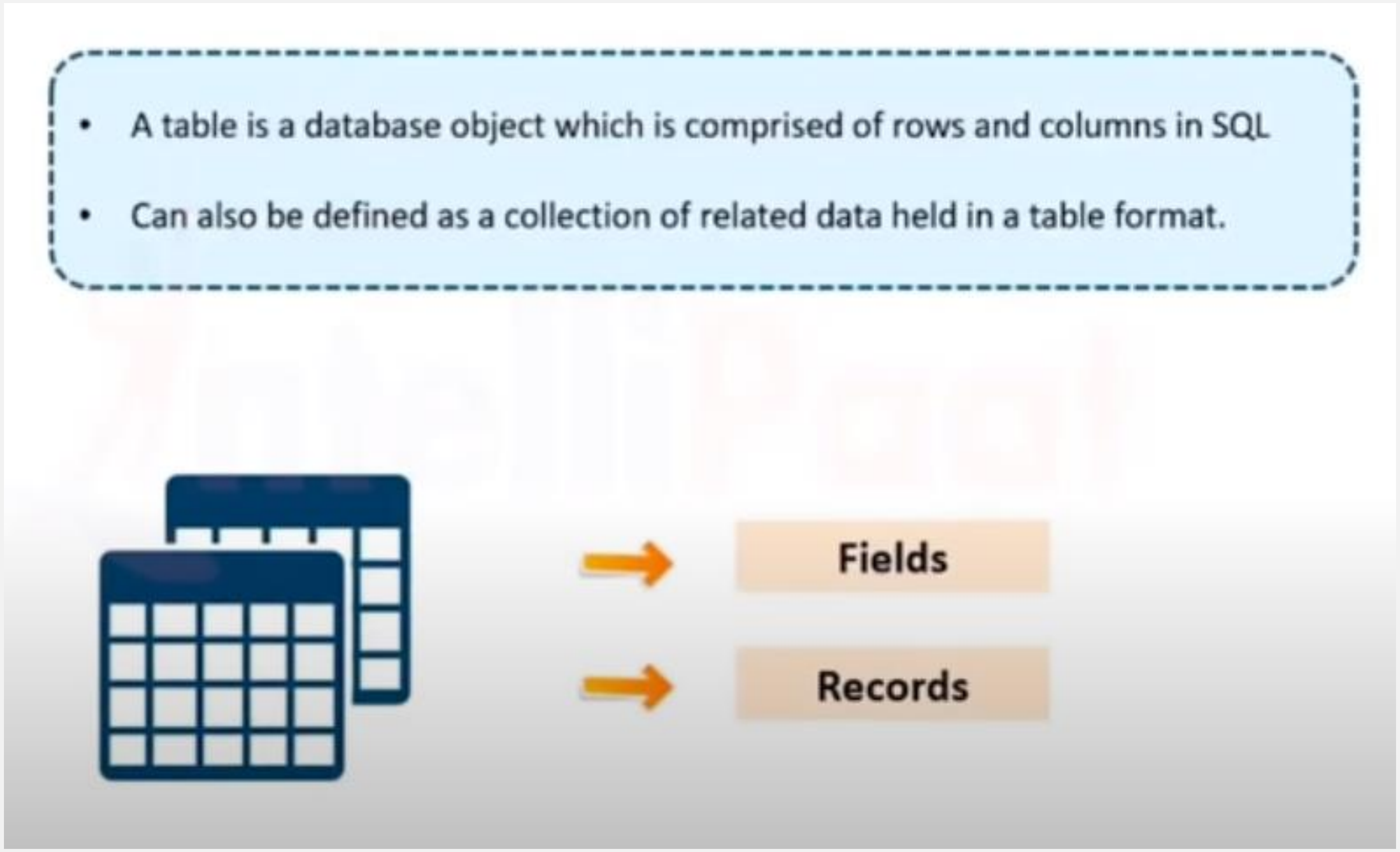
TIME – Stores the time in a HH:MM:SS format.

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.

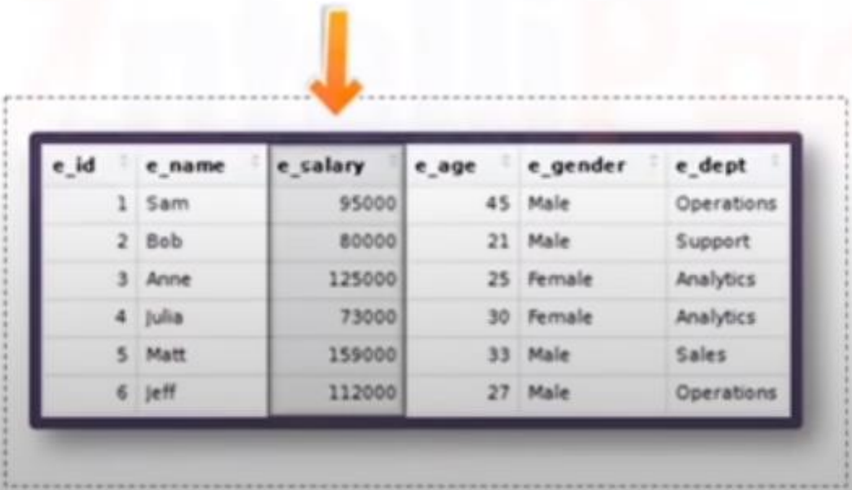
Fields

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



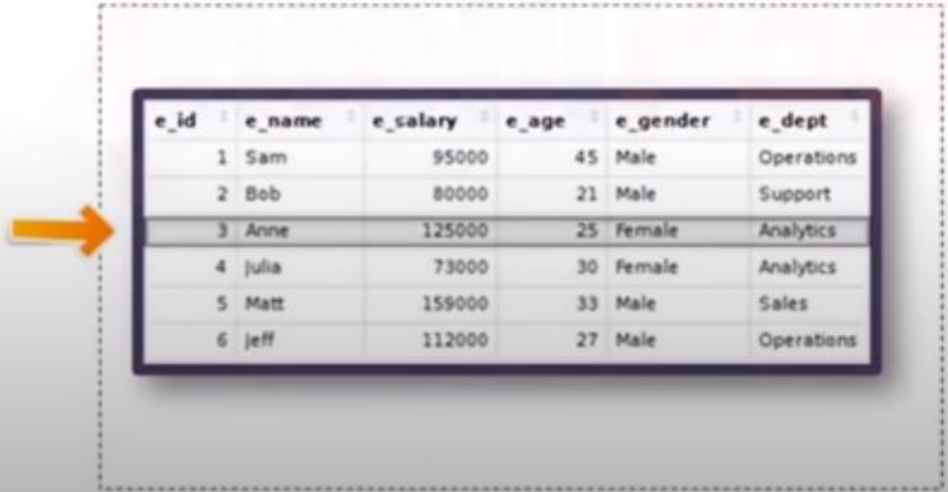
e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

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.

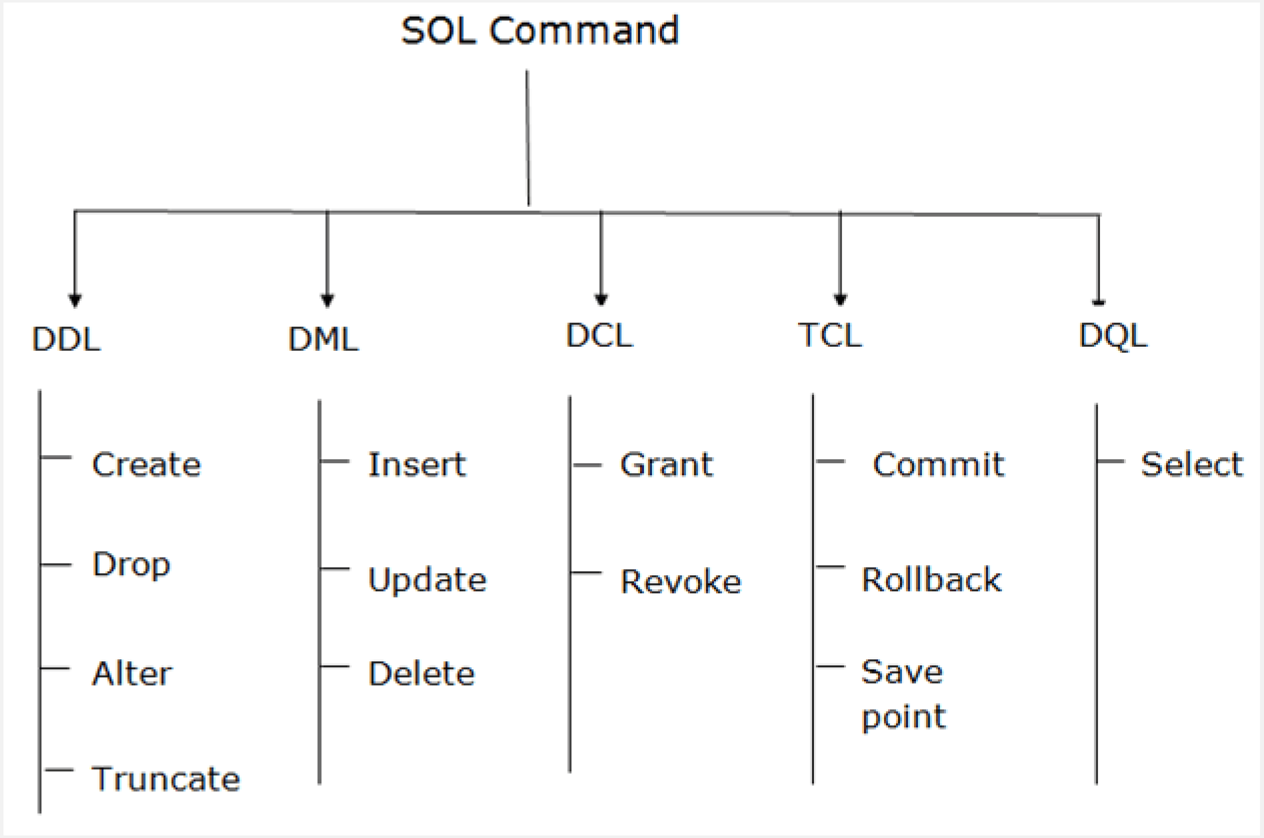
Built-in schemas in SQL Server

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

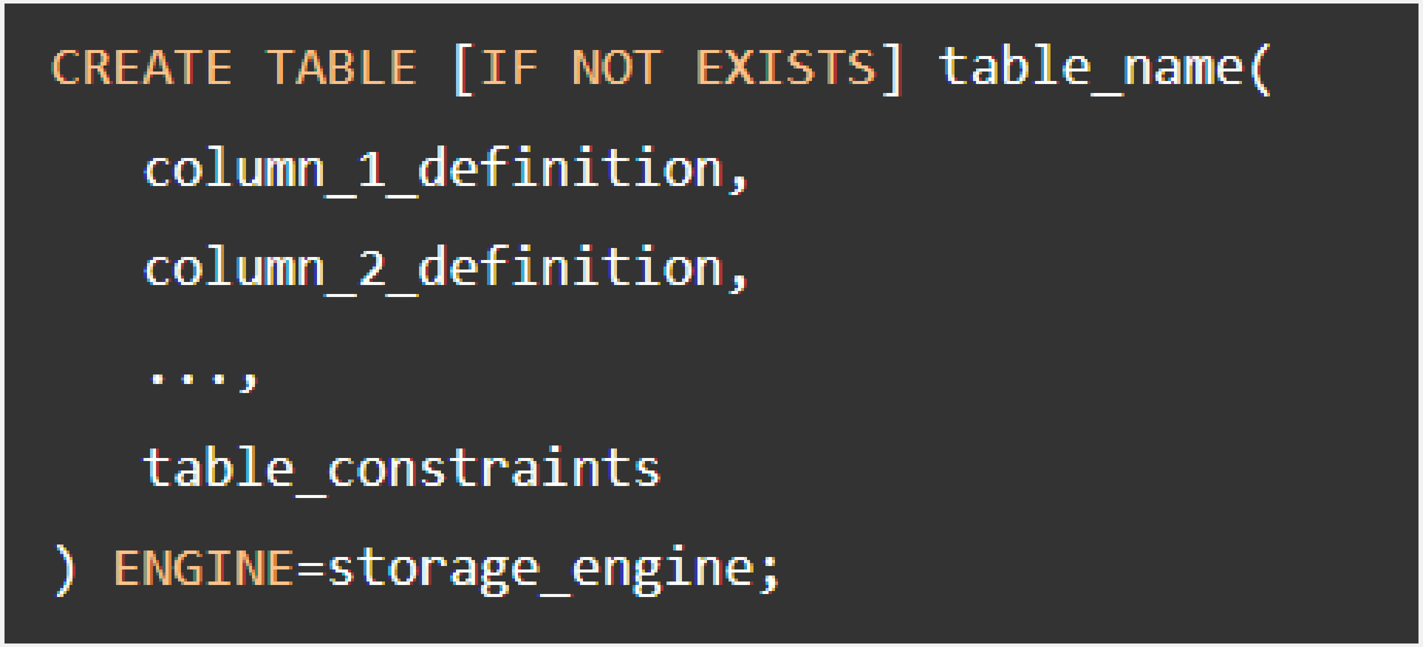
dbo, guest, sys, and INFORMATION_SCHEMA.

```
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

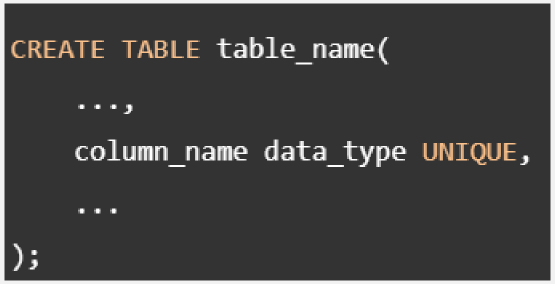
Not Null Constraint

NOT NULL Constraint

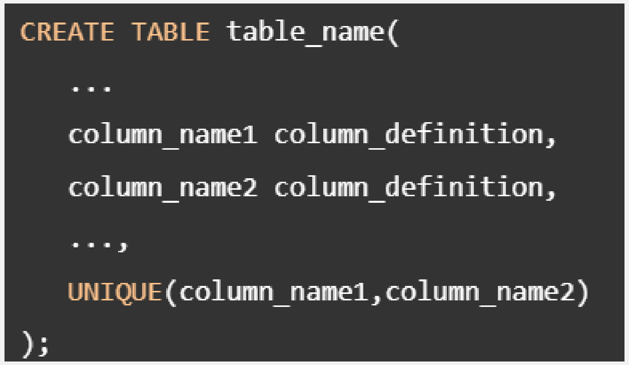
Columns in SQL Server by default store NULL values. We can restrict NULL value from being inserted into a given column by using a NOT NULL constraint.

Unique Constraint

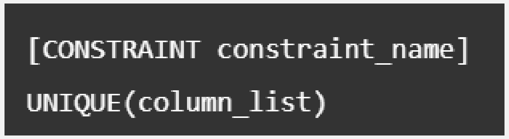
The UNIQUE constraint ensures that no duplicate values can be inserted into a column or combination of columns that are not part of the PRIMARY KEY and are participating in the UNIQUE constraint. This constraint always inserts unique and non-repetitive values into the column. It is similar to the primary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



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



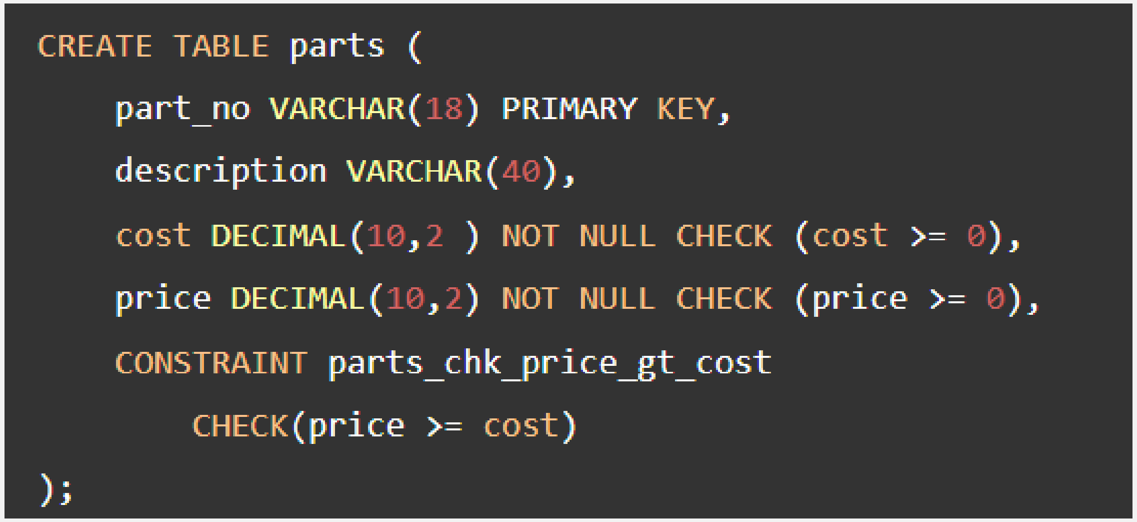
```
CREATE TABLE table_name(
   ...
   column_name1 column_definition,
   column_name2 column_definition,
   ...,
   UNIQUE(column_name1,column_name2)
);
```



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

Check Constraint

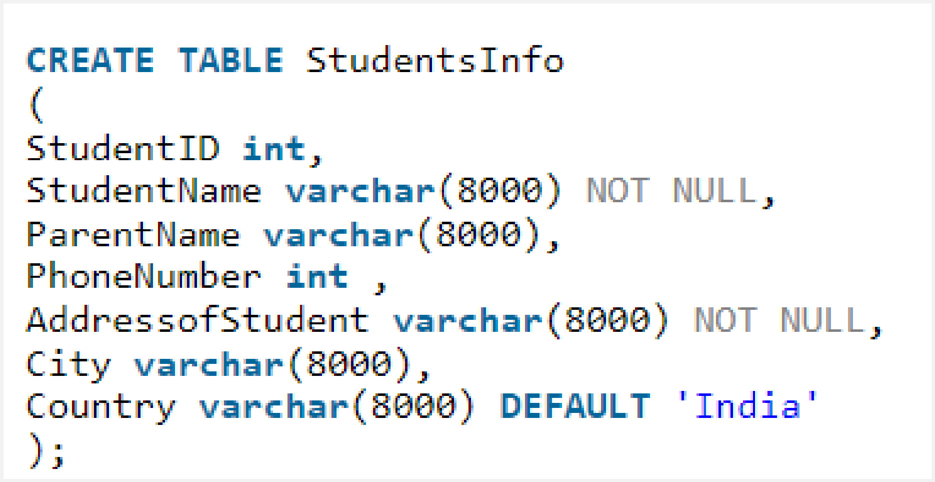
This constraint is used to limit the range of values in a column. It ensures that all the inserted values in a column must follow the specific rule.



```
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

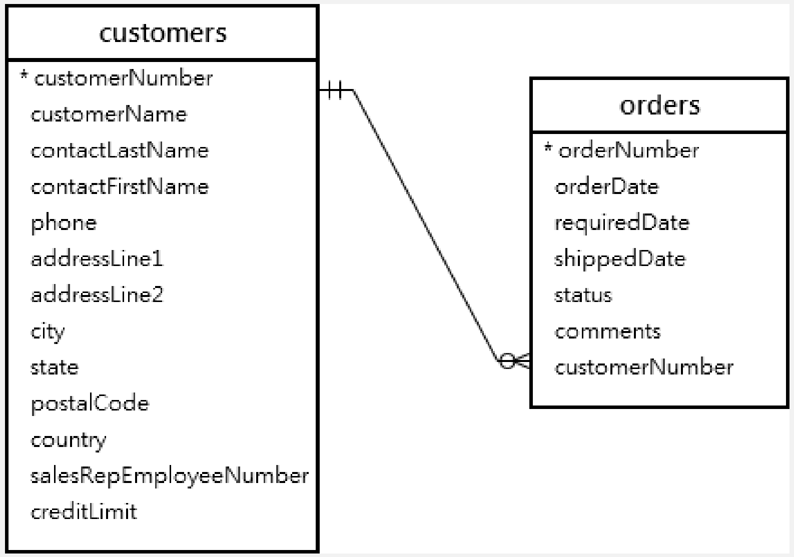
A foreign key is a database key that links two tables together. This constraint is also known as referencing key as it identifies the relationships between the tables by referencing a column of the child table containing the foreign key to the PRIMARY KEY column of the parent table. It means the foreign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

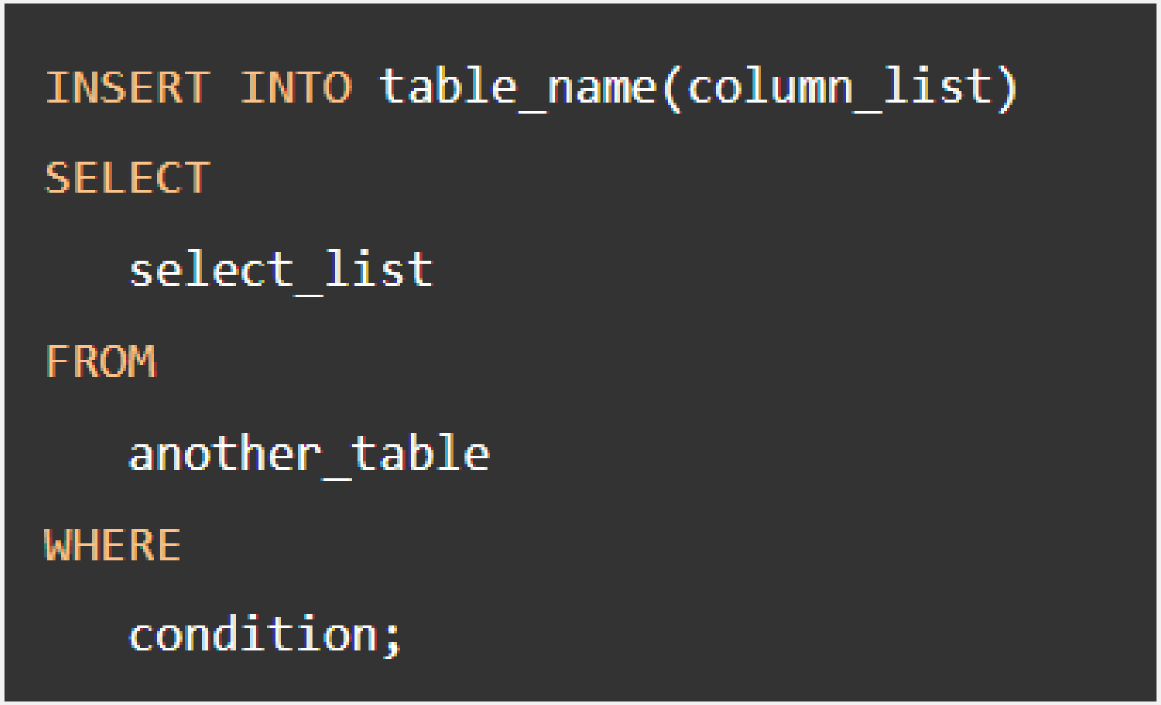
Insert Into

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

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

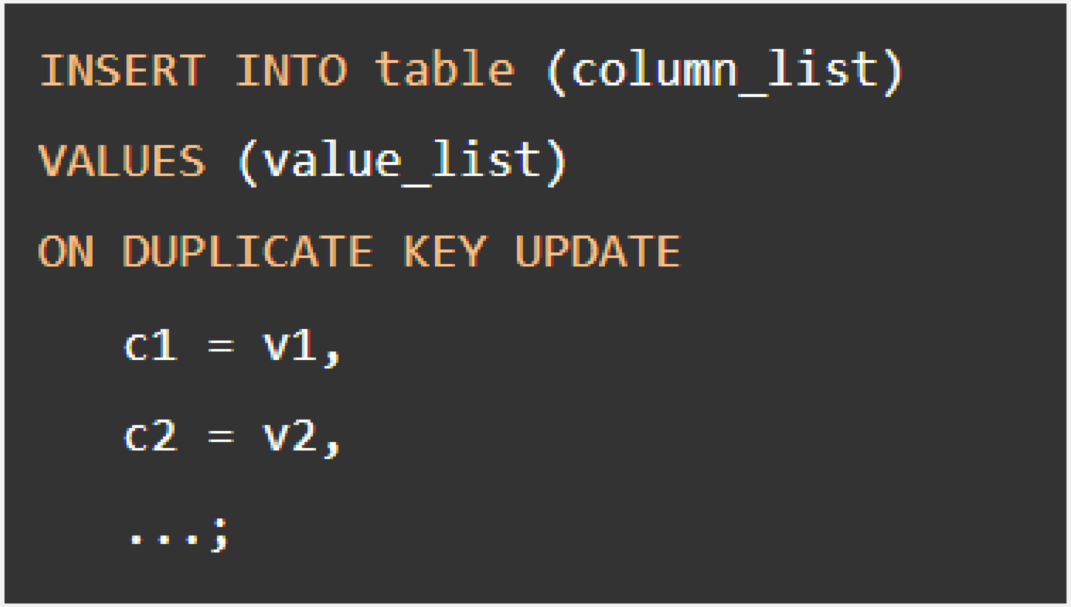
```
INSERT INTO table_name (column_list)
VALUES
        (value_list_1),
        (value_list_2),
        ...
        (value_list_n);
```

Insert Into Select



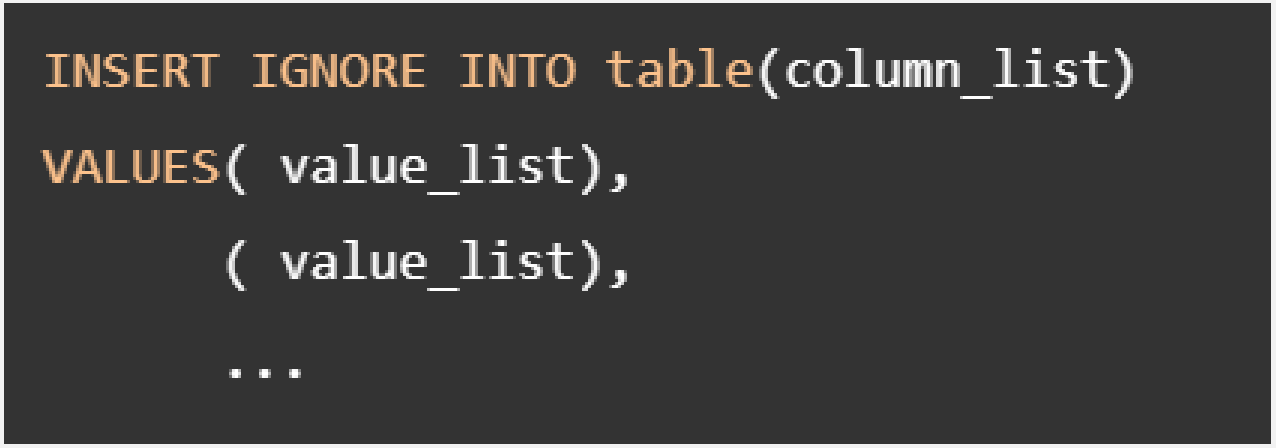
```
INSERT INTO table_name(column_list)
SELECT
    select_list
FROM
    another_table
WHERE
    condition;
```

Insert On Duplicate Key Update



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

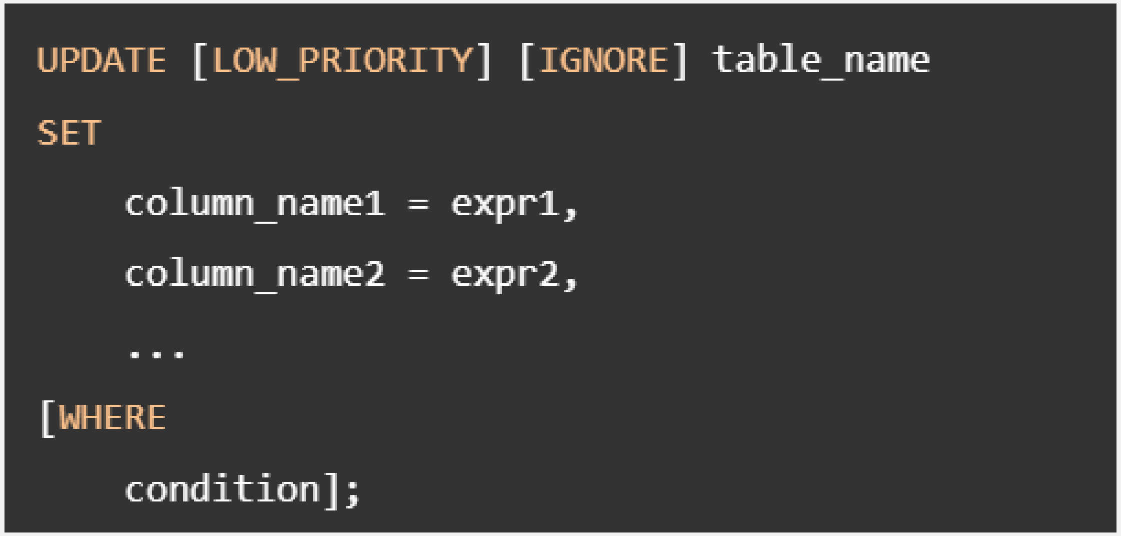
Insert Ignore



```
INSERT IGNORE INTO table(column_list)
VALUES( value_list),
      ( value_list),
      ...
```

Update

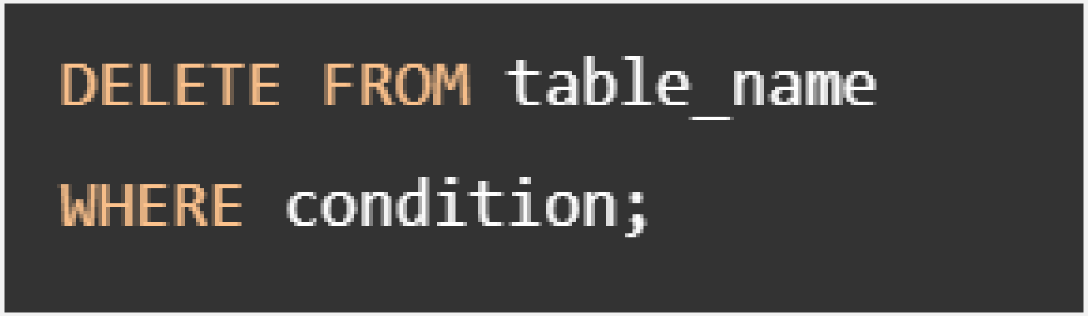
The UPDATE statement is used to modify or update the records already present in the table.



```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
[WHERE
    condition];
```

Delete

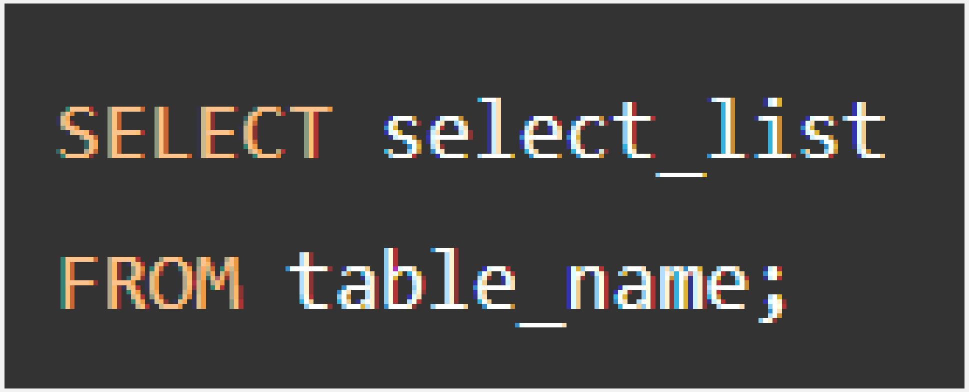
The DELETE statement is used to delete the existing records in a table.



```
DELETE FROM table_name
WHERE condition;
```

Select Statement

- Use the SELECT statement to select data from a table.
- Use the SELECT * to select data from all columns of a table.



```
SELECT select_list
FROM table_name;
```

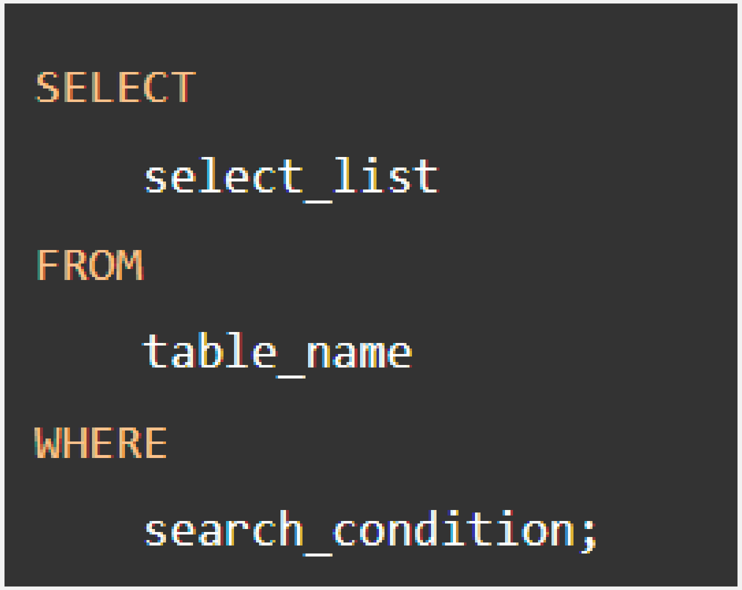
Order By

- Use the ORDER BY clause to sort the result set by one or more columns.
- Use the ASC option to sort the result set in ascending order and the DESC option to sort the result set in descending order.
- The ORDER BY clause is evaluated after the FROM and SELECT clauses.
- In MySQL, NULL is lower than non-NULL values

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column1 [ASC|DESC],
   column2 [ASC|DESC],
   ...;
```

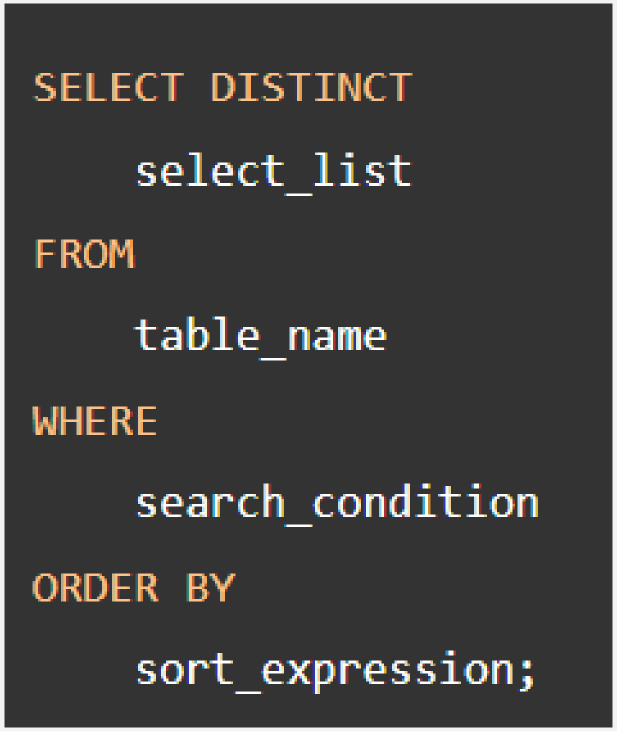
Where Clause

- Use the WHERE clause to filter rows by a condition.
- MySQL evaluates the WHERE clause after the FROM clause and before the SELECT and ORDER BY clauses.



```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

Select Distinct



```
SELECT DISTINCT
    select_list
FROM
    table_name
WHERE
    search_condition
ORDER BY
    sort_expression;
```

Use the MySQL DISTINCT clause to remove duplicate rows from the result set returned by the SELECT clause.

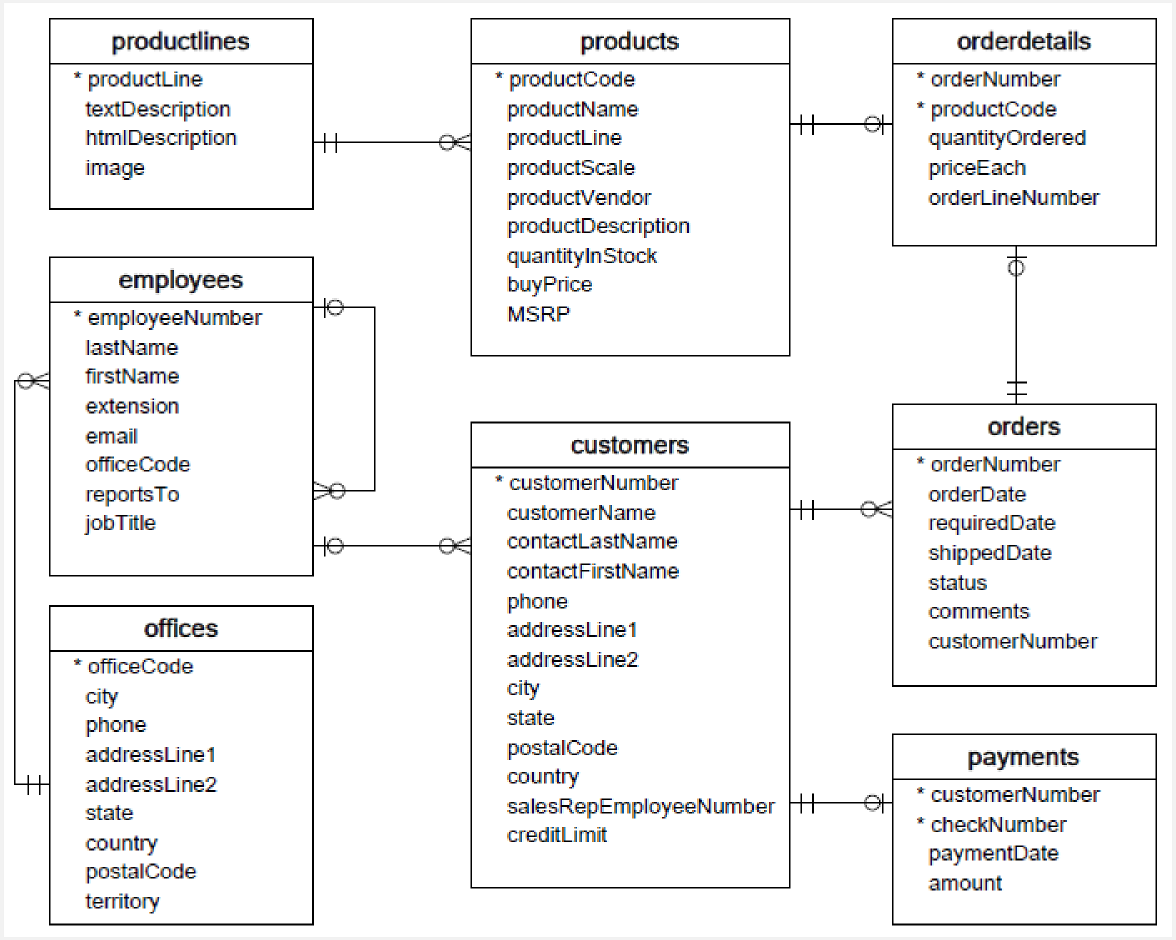
Relational Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Sample Database

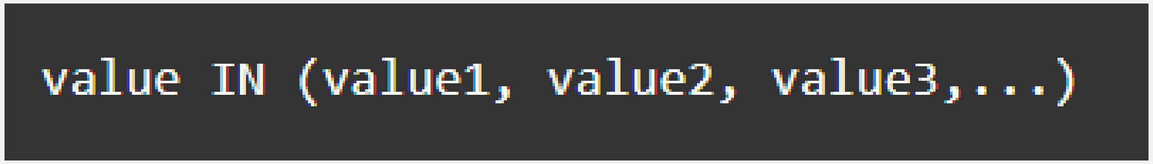


Logical Operator Operations

- Find customers from USA and state CA
- Find customers from USA and state CA whose creditlimit > 100000
- Find customers from USA or France
- Find customers from USA or Franc whose creditlimit > 100000

In Operator

- Use the IN operator to check if a value is in a set of values.
- Use the IN operator to form a condition for the WHERE clause.



```
value IN (value1, value2, value3,...)
```

Between Operator

- Use the MySQL BETWEEN operator to test if a value falls within a range of values.

```
value BETWEEN low AND high;
```

To negate the BETWEEN operator, you use the NOT operator:

```
value NOT BETWEEN low AND high
```

Between Operator Operations

- Find product information from Products where buyPrice is between 90 and 100 [try with not]
- Find order information from Orders between dates 1/1/2003 to 31/1/2003

Like Operator

- Use the LIKE operator to test if a value matches a pattern.
- The % wildcard matches zero or more characters.
- The _ wildcard matches a single character.
- Use ESCAPE clause specifies an escape character other than the default escape character (\).
- Use the NOT operator to negate the LIKE operator.

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"

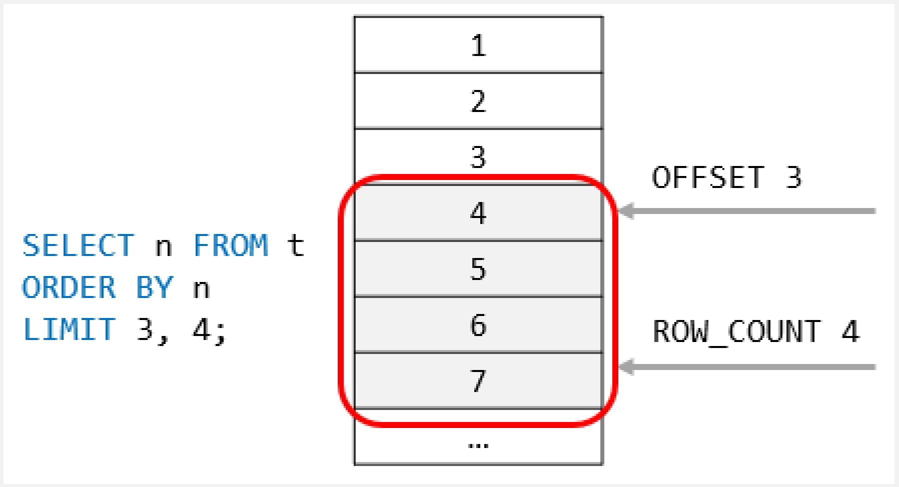
Like Operator Operations

- Find employees whose first name starts with “a”
- Find employees whose last name ends “on”
- Find employees whose last name contains “on”
- Find employees whose first name like “Tim” , “Tom” etc.
- Find employees whose last names don’t start with the letter B

Select Limit

- Use the MySQL LIMIT clause to constrain the number of rows returned by the SELECT statement.

```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
```



```
SELECT n FROM t
ORDER BY n
LIMIT 3, 4;
```

Limit Operations

- Get the top five customers who have the highest credit
- Find customer who have second highest credit
- Find first five unique states from customers

Is Null

To test whether a value is NULL or not, you use the IS NULL operator.

```
value IS NULL
```

```
value IS NOT NULL
```

Is Null Operations

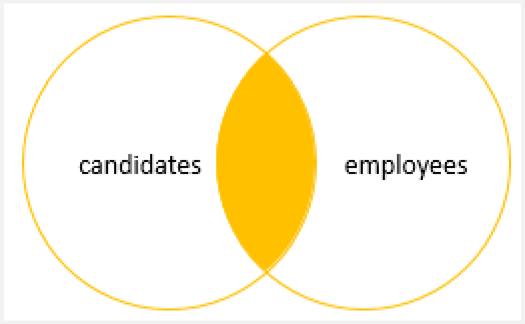
- Find customers who do not have a sales representative
- Find customers who do have a sales representative

Joins

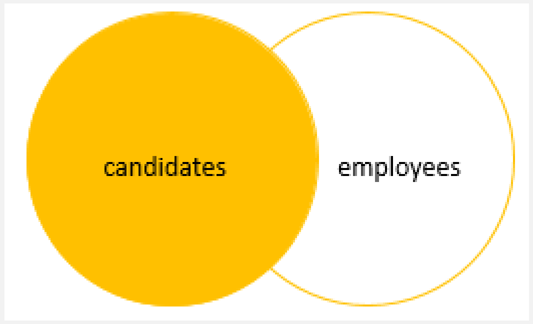
Joins are used to combine tuples from two or more tables, based on a related column between the tables.

- Inner join
- Left join
- Right join
- Cross join

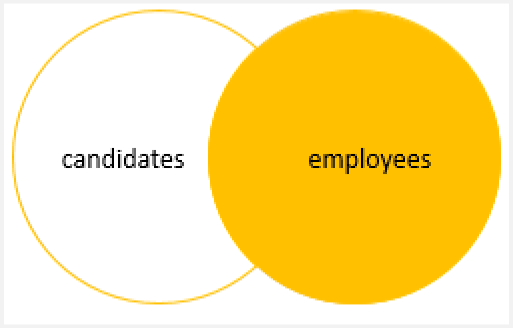
Joins



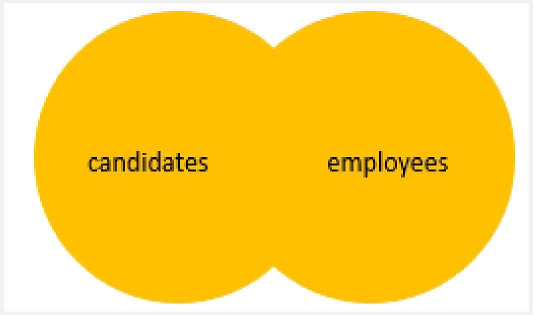
Inner Join



Left Join



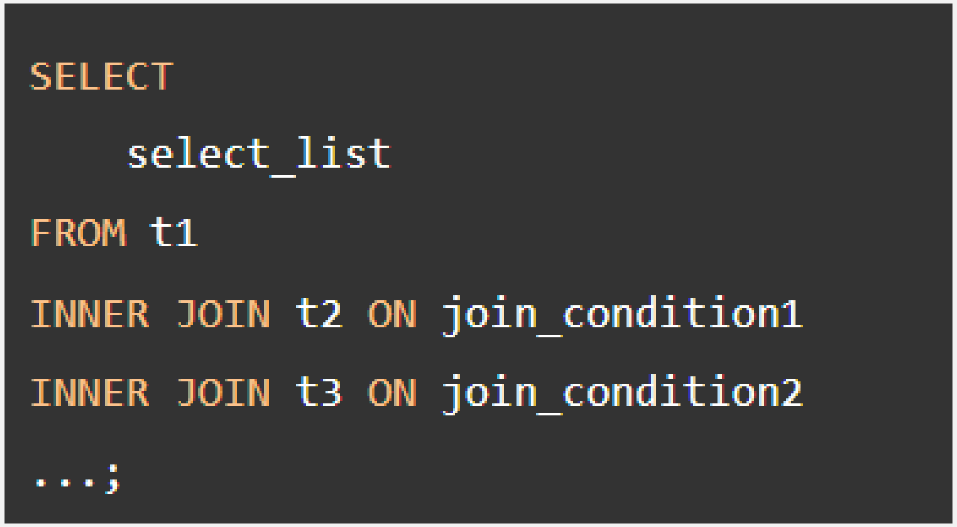
Right Join



Full Join

Inner Join

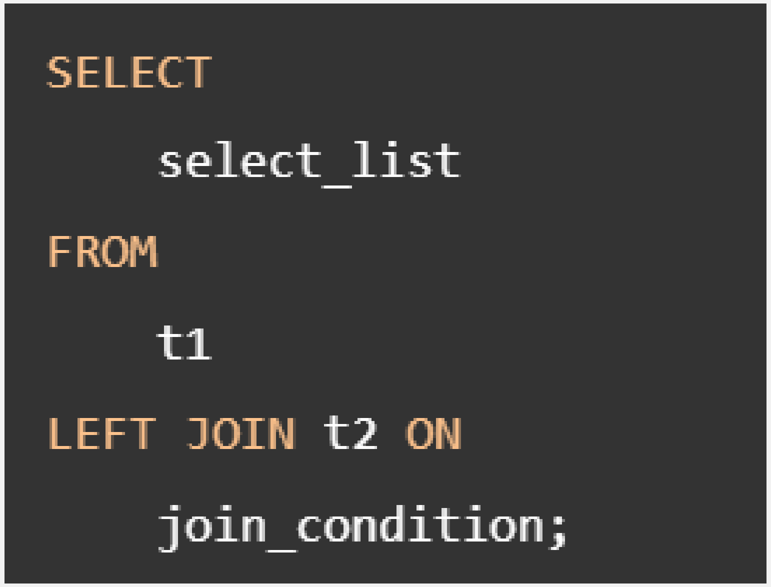
The INNER JOIN matches each row in one table with every row in other tables and allows you to query rows that contain columns from both tables.



```
SELECT
    select_list
FROM t1
INNER JOIN t2 ON join_condition1
INNER JOIN t3 ON join_condition2
...;
```

Left Join

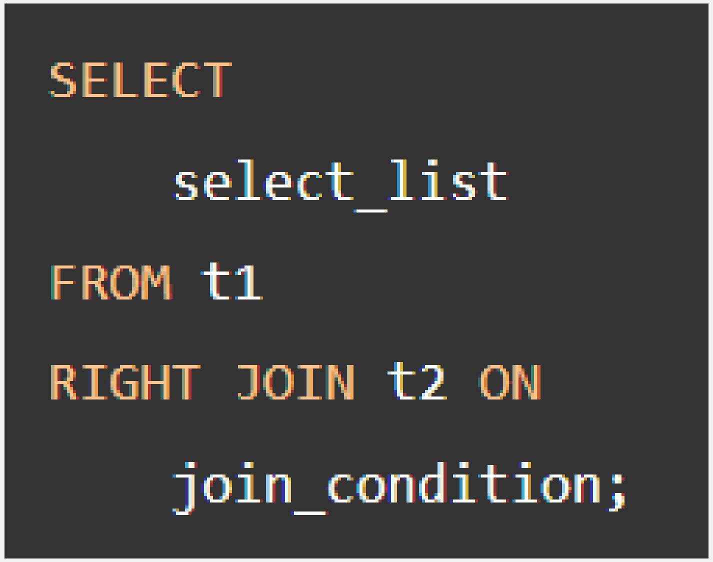
The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



```
SELECT
    select_list
FROM
    t1
LEFT JOIN t2 ON
    join_condition;
```

Right Join

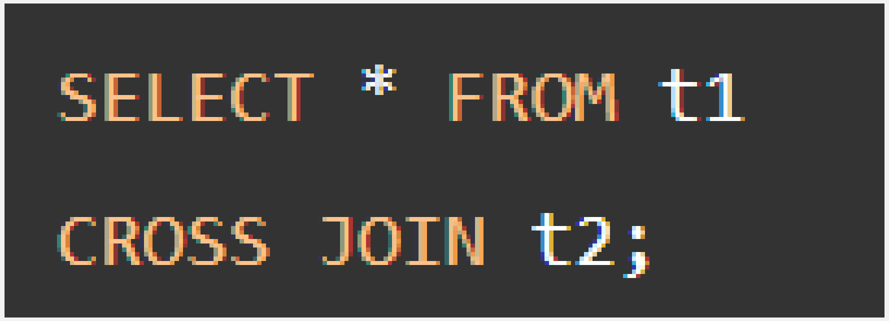
The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



```
SELECT
    select_list
FROM t1
RIGHT JOIN t2 ON
    join_condition;
```

Cross Join

The CROSS JOIN keyword returns all records from both tables (table1 and table2).

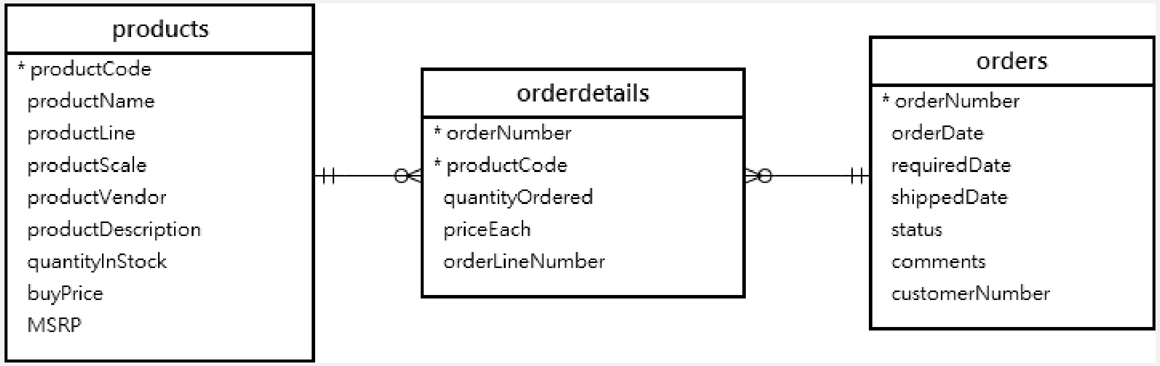


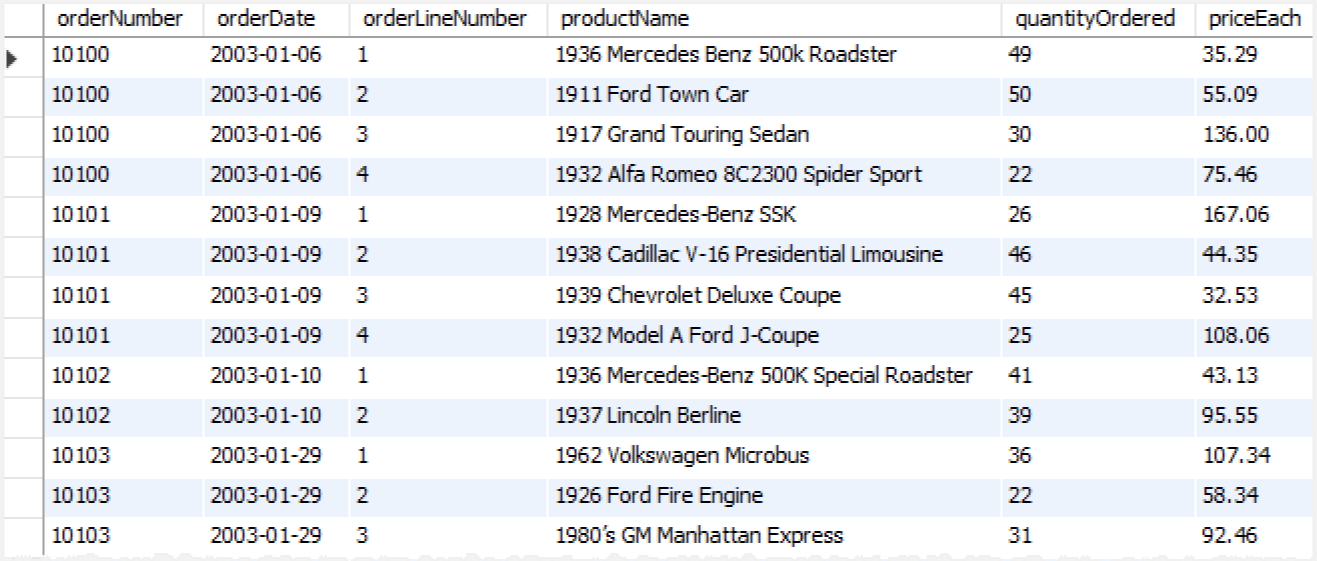
```
SELECT * FROM t1
CROSS JOIN t2;
```

Join Operations

- Find The productCode and productName from the products table and textDescription of product lines from the productlines table. *[To do this, you need to select data from both tables by matching rows based on values in the productline column using the INNER JOIN clause]*

Join Operations

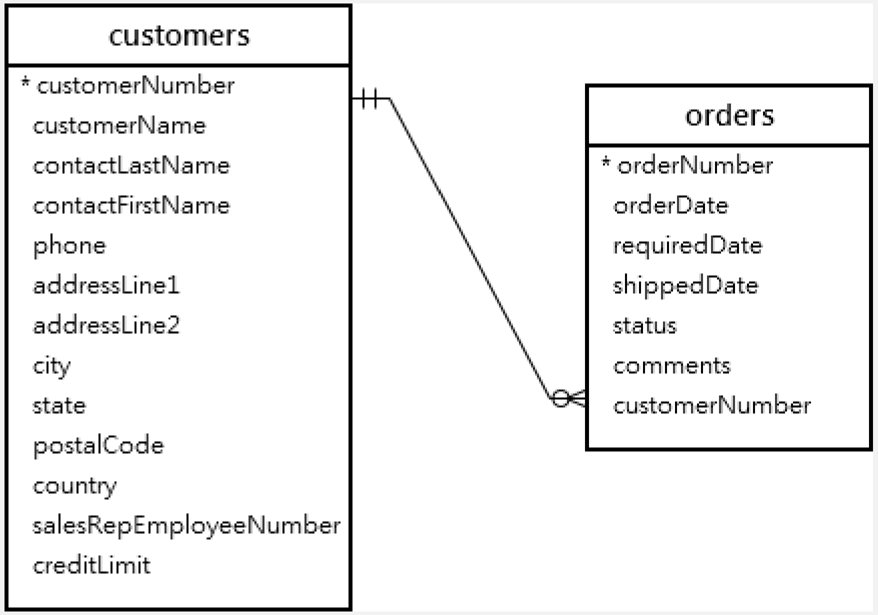




orderNumber	orderDate	orderLineNumber	productName	quantityOrdered	priceEach
10100	2003-01-06	1	1936 Mercedes Benz 500k Roadster	49	35.29
10100	2003-01-06	2	1911 Ford Town Car	50	55.09
10100	2003-01-06	3	1917 Grand Touring Sedan	30	136.00
10100	2003-01-06	4	1932 Alfa Romeo 8C2300 Spider Sport	22	75.46
10101	2003-01-09	1	1928 Mercedes-Benz SSK	26	167.06
10101	2003-01-09	2	1938 Cadillac V-16 Presidential Limousine	46	44.35
10101	2003-01-09	3	1939 Chevrolet Deluxe Coupe	45	32.53
10101	2003-01-09	4	1932 Model A Ford J-Coupe	25	108.06
10102	2003-01-10	1	1936 Mercedes-Benz 500K Special Roadster	41	43.13
10102	2003-01-10	2	1937 Lincoln Berline	39	95.55
10103	2003-01-29	1	1962 Volkswagen Microbus	36	107.34
10103	2003-01-29	2	1926 Ford Fire Engine	22	58.34
10103	2003-01-29	3	1980's GM Manhattan Express	31	92.46

Join Operations

- Find all customers and their orders
- Find all customer who have no orders



Join Operations

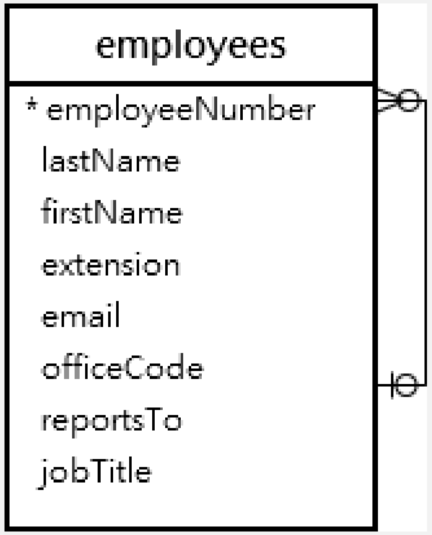
Type Following query and Justify Answer

```
SELECT
    o.orderNumber,
    customerNumber,
    productCode
FROM
    orders o
LEFT JOIN orderDetails
    USING (orderNumber)
WHERE
    orderNumber = 10123;
```

```
SELECT
    o.orderNumber,
    customerNumber,
    productCode
FROM
    orders o
LEFT JOIN orderDetails d
    ON o.orderNumber = d.orderNumber AND
       o.orderNumber = 10123;
```

Self Join

A self join is a regular join, but the table is joined with itself.

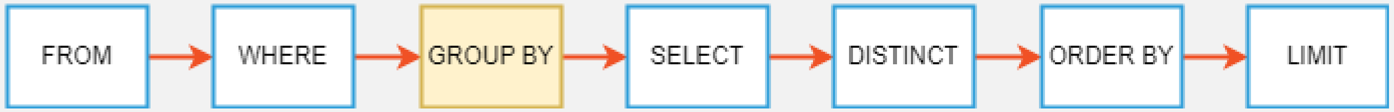


To get the whole organization structure, you can join the employees table to itself using the employeeNumber and reportsTo columns. The table employees has two roles: one is the Manager and the other is Direct Reports.

Group By Clause

Use the GROUP BY clause to group rows into subgroups.

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

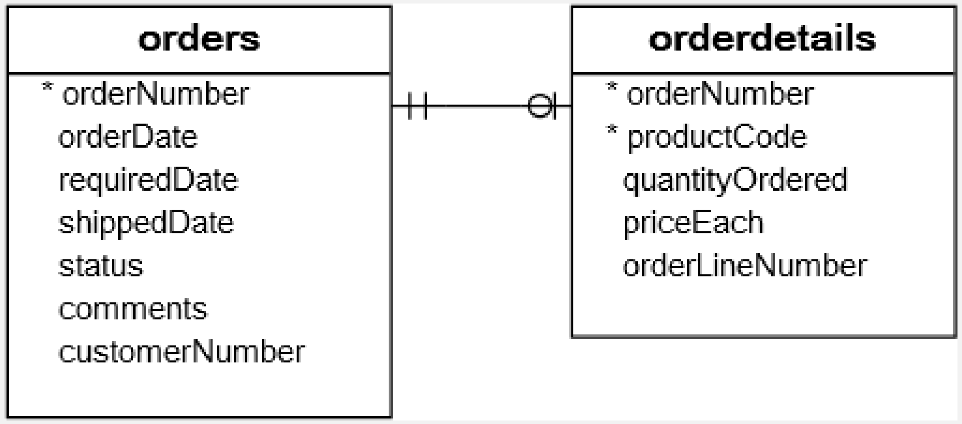


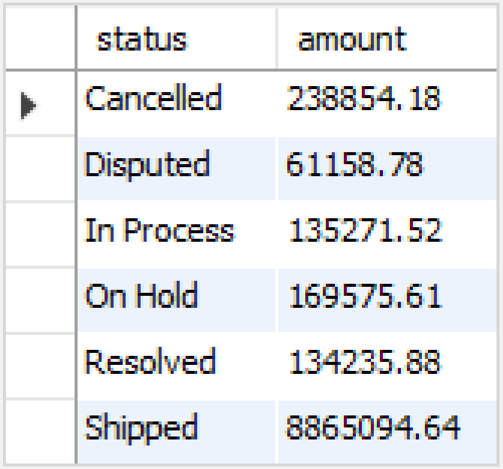
Aggregate Functions

Aggregate function	Description
AVG()	Return the average of non-NULL values.
BIT_AND()	Return bitwise AND.
BIT_OR()	Return bitwise OR.
BIT_XOR()	Return bitwise XOR.
COUNT()	Return the number of rows in a group, including rows with NULL values.
GROUP_CONCAT()	Return a concatenated string.
JSON_ARRAYAGG()	Return result set as a single JSON array.
JSON_OBJECTAGG()	Return result set as a single JSON object.
MAX()	Return the highest value (maximum) in a set of non-NULL values.
MIN()	Return the lowest value (minimum) in a set of non-NULL values.
STDEV()	Return the population standard deviation.
SUM()	Return the summation of all non-NULL values a set.
VARIANCE()	Return the population standard variance.

Group By Operations

- Find the number of orders in each status from Orders Table
- Find – Amount = sum(quantityOrdered * priceEach)

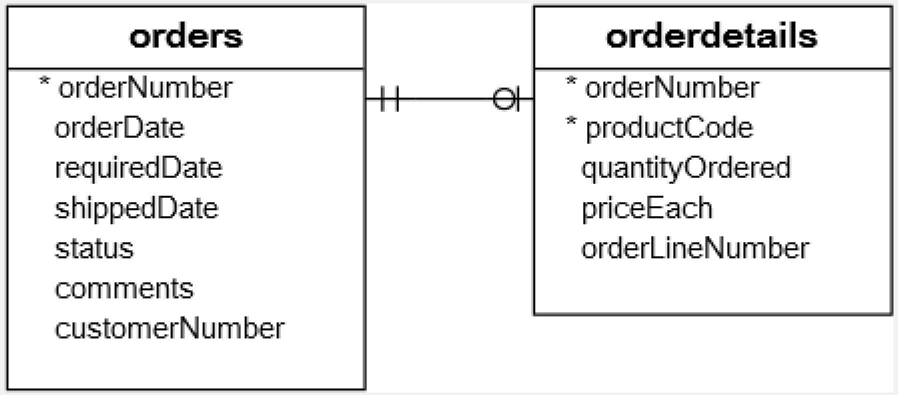


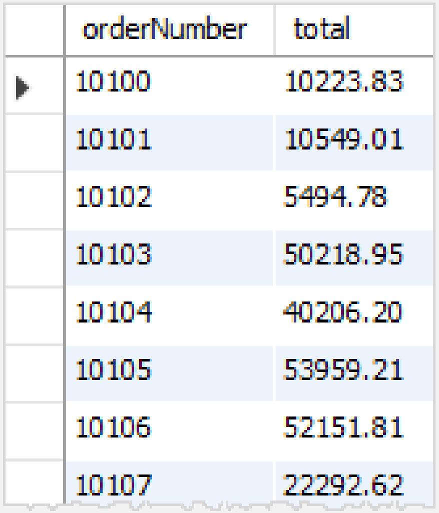


status	amount
Cancelled	238854.18
Disputed	61158.78
In Process	135271.52
On Hold	169575.61
Resolved	134235.88
Shipped	8865094.64

Group By Operations

- Find – Total = sum(quantityOrdered * priceEach)

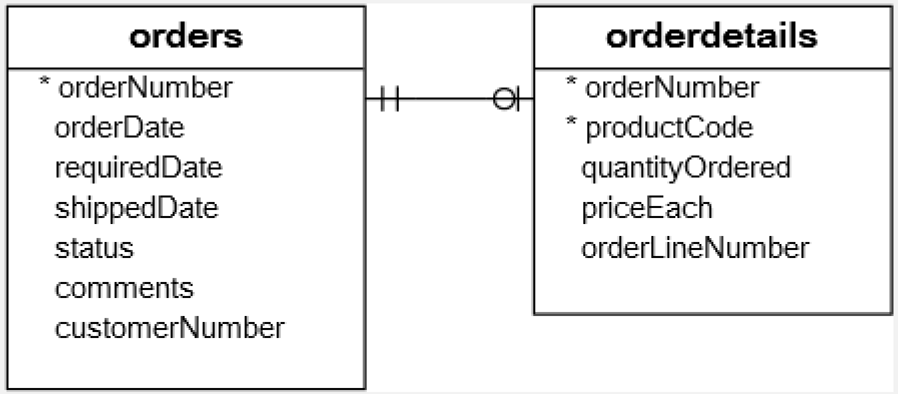


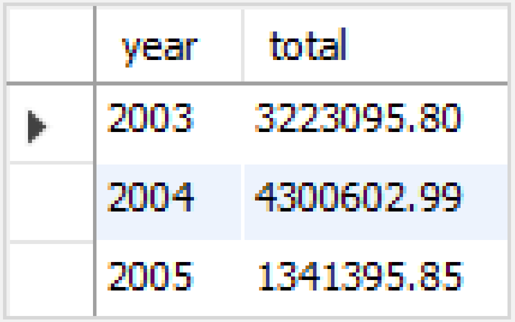


orderNumber	total
10100	10223.83
10101	10549.01
10102	5494.78
10103	50218.95
10104	40206.20
10105	53959.21
10106	52151.81
10107	22292.62

Group By Operations

- Find – Total = sum(quantityOrdered * priceEach)
  for status = “Shipped”
  Year (orderDate)

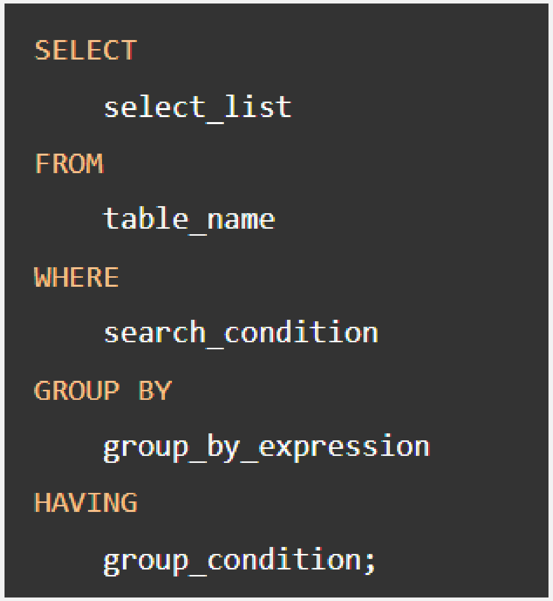




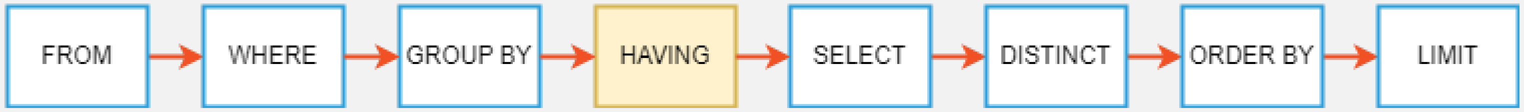
year	total
2003	3223095.80
2004	4300602.99
2005	1341395.85

Group By with having clause

- The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified condition. If you omit the GROUP BY clause, the HAVING clause behaves like the WHERE clause.

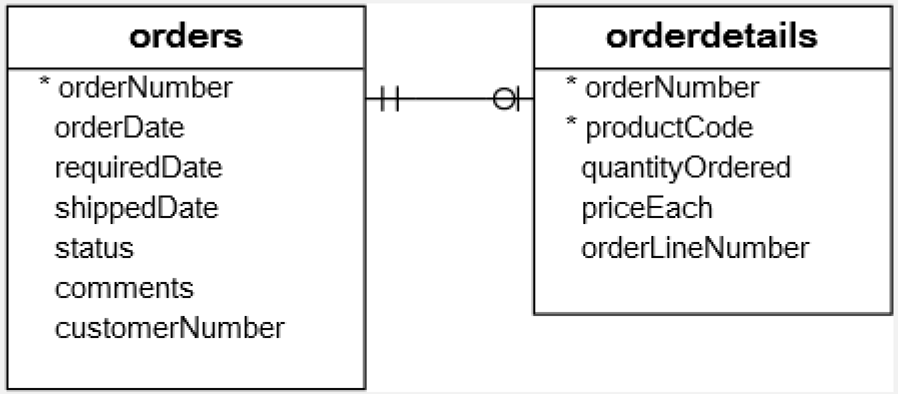


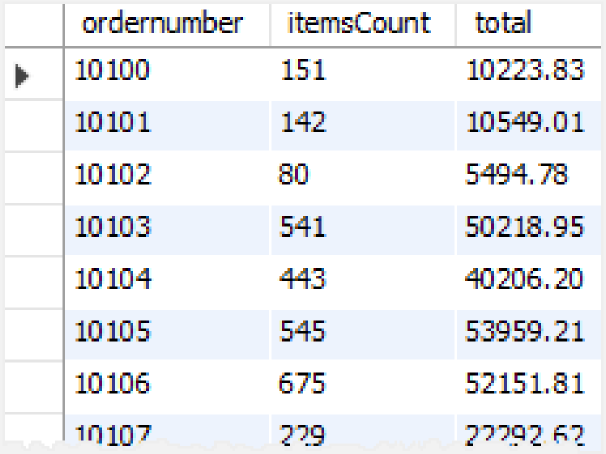
```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
GROUP BY
    group_by_expression
HAVING
    group_condition;
```



Group By With Having Operations

- Find – Total = sum(quantityOrdered * priceEach)
  ItemCount = sum (quantityOrdered)
  having total > 1000

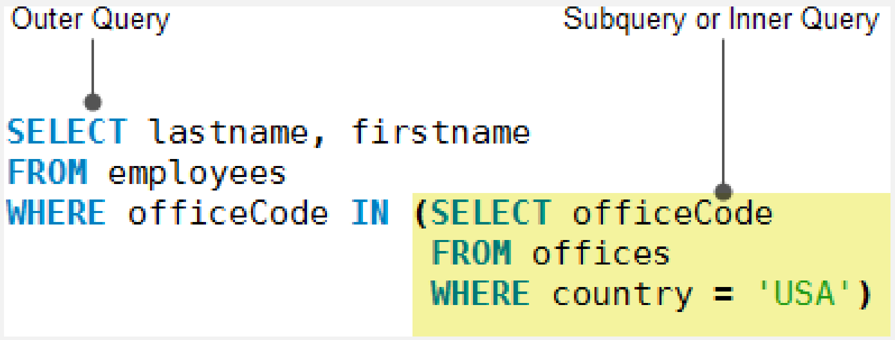




ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

Subquery

A subquery is a query nested within another query (or outer query).



```
SELECT lastname, firstname
FROM employees
WHERE officeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

- The subquery returns all office codes of the offices located in the USA.
- The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.

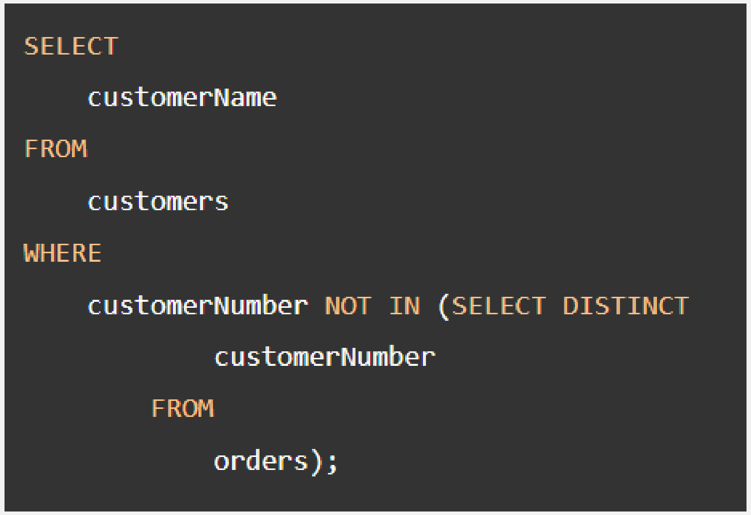
Subquery Operations

- Find Customer having highest payment from payments (hint : where amount = select max (amount) from payments)
- Find customers whose payments are greater than the average payment using a subquery:

Subquery Operations

- If a subquery returns more than one value, you can use other operators such as IN or NOT IN operator in the WHERE clause.

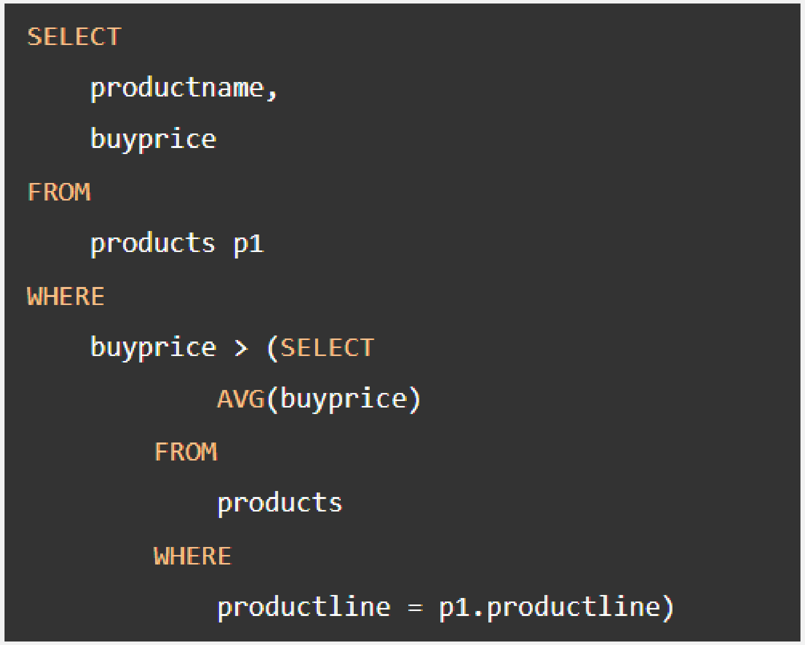
- For example, you can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows:



```
SELECT
    customerName
FROM
    customers
WHERE
    customerNumber NOT IN (SELECT DISTINCT
            customerNumber
        FROM
            orders);
```

Correlated Subquery Operations

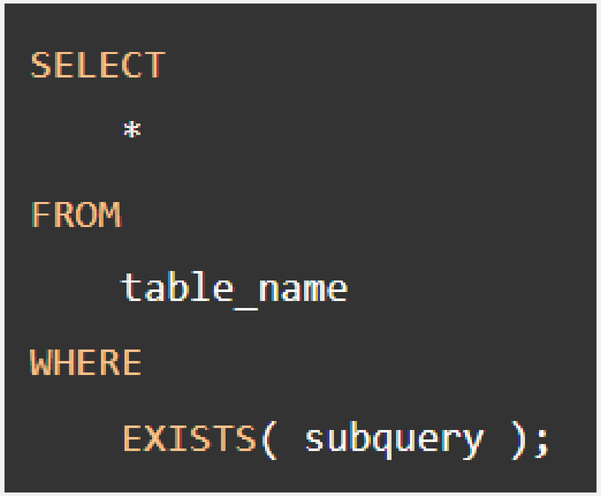
- Unlike a standalone subquery, a correlated subquery is a subquery that uses the data from the outer query. In other words, a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.



```
SELECT
    productname,
    buyprice
FROM
    products p1
WHERE
    buyprice > (SELECT
            AVG(buyprice)
        FROM
            products
        WHERE
            productline = p1.productline)
```

Exists and Not Exists in Subquery

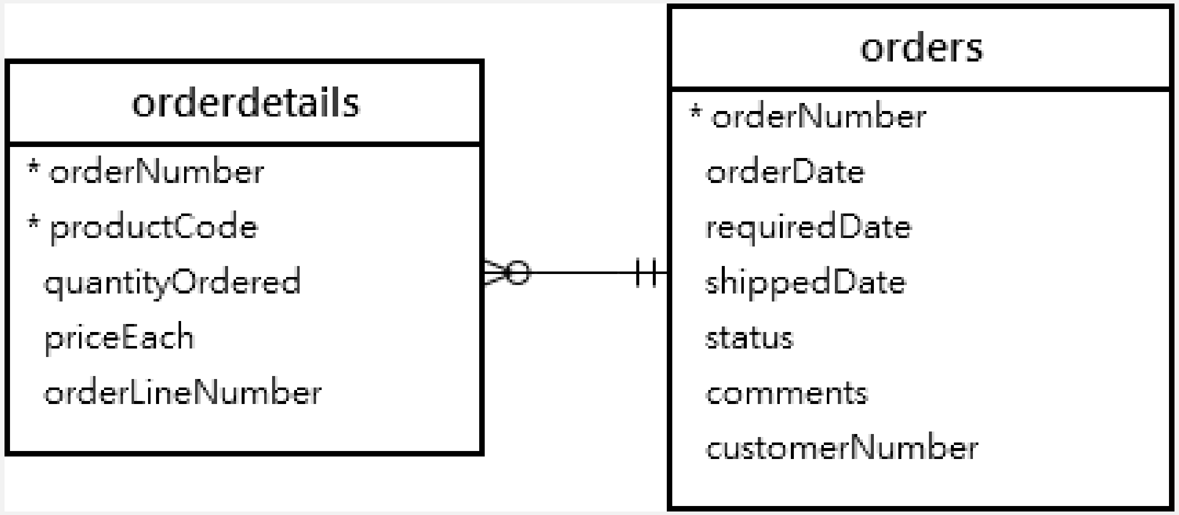
- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.



```
SELECT
    *
FROM
    table_name
WHERE
    EXISTS( subquery );
```

Exists and Not Exists in Subquery Operations

- Find customers who placed at least one sales order with the total value greater than 60K by using the EXISTS operator



Any Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

- ANY means that the condition will be true if the operation is true for any of the values in the range

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

All Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
  is used with SELECT, WHERE and HAVING statements

- ALL means that the condition will be true only if the operation is true for all values in the range.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Any and All Operations

"Products" table

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97

Any and All Operations

"Order Details" table

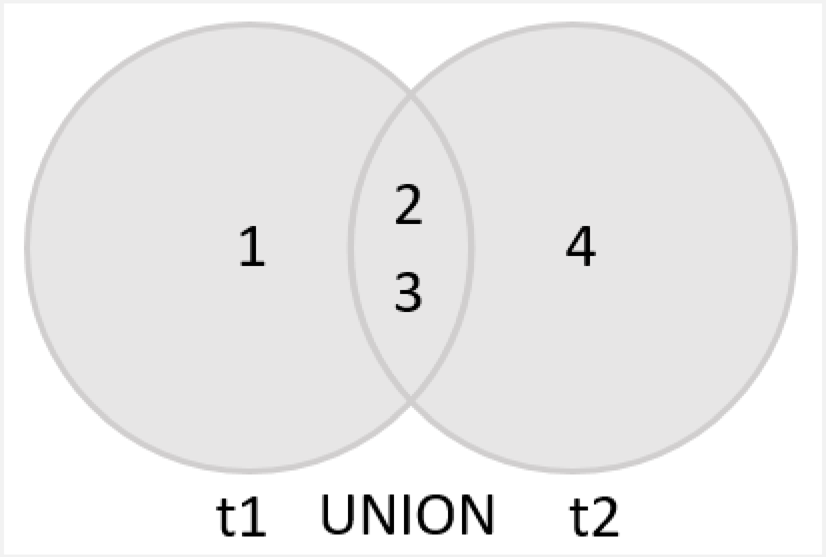
OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40
6	10250	41	10
7	10250	51	35
8	10250	65	15
9	10251	22	6
10	10251	57	15

Any and All Operations

- lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99
- lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10
- lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10.

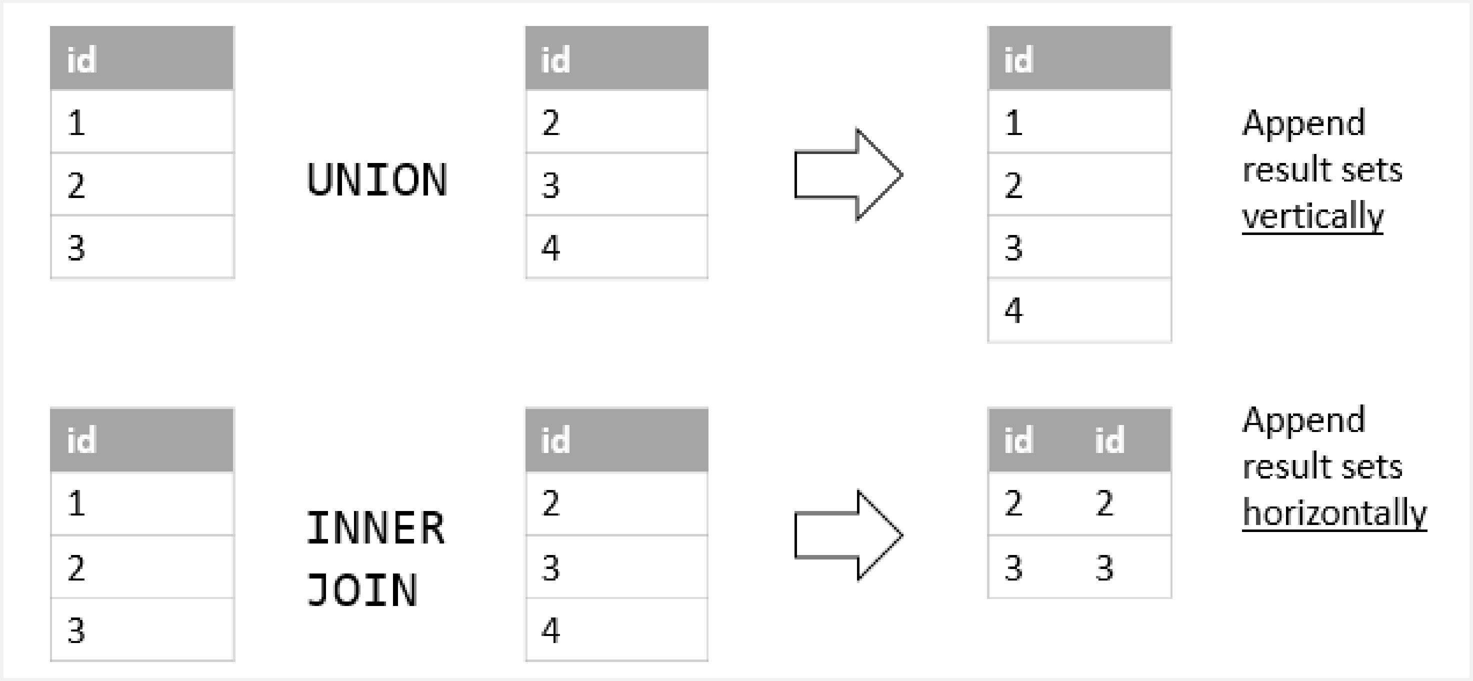
Union Operator

- MySQL UNION operator allows you to combine two or more result sets of queries into a single result set



```
SELECT column_list
UNION [DISTINCT | ALL]
SELECT column_list
UNION [DISTINCT | ALL]
SELECT column_list
...
```

Union Vs. Join



id
1
2
3

UNION

id
2
3
4

id
1
2
3
4

Append result sets vertically

id
1
2
3

INNER JOIN

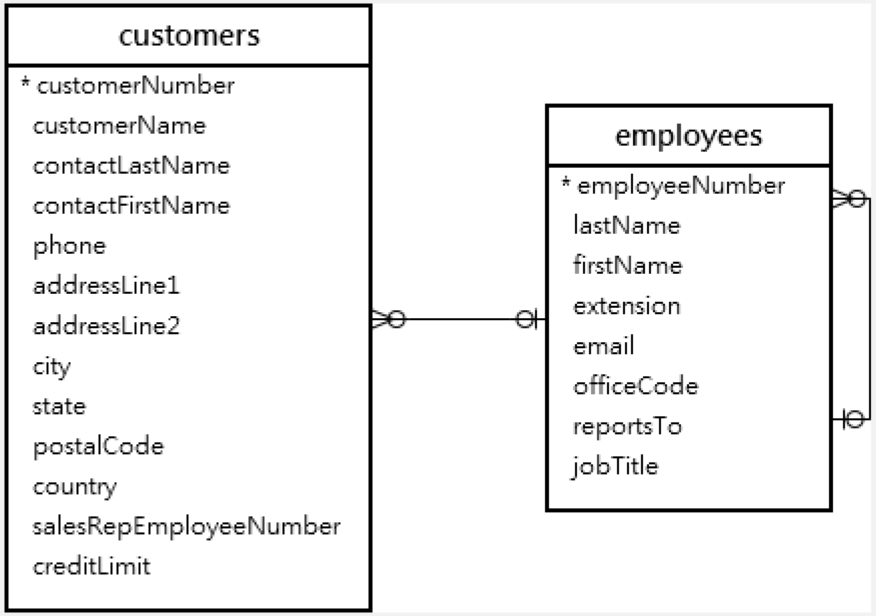
id
2
3
4

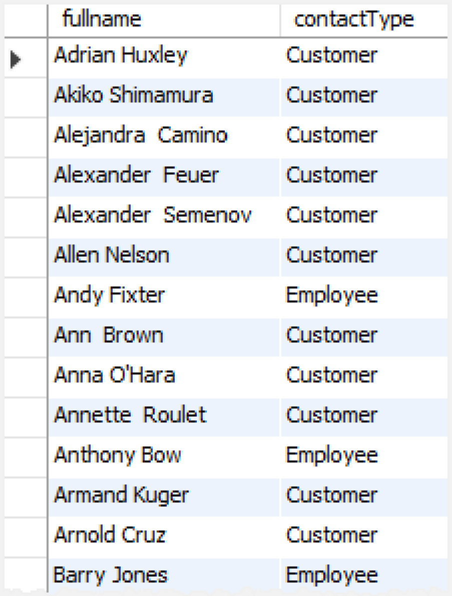
id	id
2	2
3	3

Append result sets horizontally

Union Operations

combine the first name and last name of employees and customers into a single result set





fullname	contactType
Adrian Huxley	Customer
Akiko Shimamura	Customer
Alejandra Camino	Customer
Alexander Feuer	Customer
Alexander Semenov	Customer
Allen Nelson	Customer
Andy Fixter	Employee
Ann Brown	Customer
Anna O'Hara	Customer
Annette Roulet	Customer
Anthony Bow	Employee
Armand Kuger	Customer
Arnold Cruz	Customer
Barry Jones	Employee

String Functions

Name	Description
CONCAT	Concatenate two or more strings into a single string
INSTR	Return the position of the first occurrence of a substring in a string
LENGTH	Get the length of a string in bytes and in characters
LEFT	Get a specified number of leftmost characters from a string
LOWER	Convert a string to lowercase
LTRIM	Remove all leading spaces from a string
REPLACE	Search and replace a substring in a string
RIGHT	Get a specified number of rightmost characters from a string
RTRIM	Remove all trailing spaces from a string
SUBSTRING	Extract a substring starting from a position with a specific length.
SUBSTRING_INDEX	Return a substring from a string before a specified number of occurrences of a delimiter
TRIM	Remove unwanted characters from a string.
FIND_IN_SET	Find a string within a comma-separated list of strings
FORMAT	Format a number with a specific locale, rounded to the number of decimals
UPPER	Convert a string to uppercase

Date Functions

Function	Description
CURDATE	Returns the current date.
DATEDIFF	Calculates the number of days between two DATE values.
DAY	Gets the day of the month of a specified date.
DATE_ADD	Adds a time value to date value.
DATE_SUB	Subtracts a time value from a date value.
DATE_FORMAT	Formats a date value based on a specified date format.
DAYNAME	Gets the name of a weekday for a specified date.
DAYOFWEEK	Returns the weekday index for a date.
EXTRACT	Extracts a part of a date.
LAST_DAY	Returns the last day of the month of a specified date
NOW	Returns the current date and time at which the statement executed.
MONTH	Returns an integer that represents a month of a specified date.
STR_TO_DATE	Converts a string into a date and time value based on a specified format.
SYSDATE	Returns the current date.
TIMEDIFF	Calculates the difference between two TIME or DATETIME values.
TIMESTAMPDIFF	Calculates the difference between two DATE or DATETIME values.
WEEK	Returns a week number of a date.
WEEKDAY	Returns a weekday index for a date.
YEAR	Return the year for a specified date

Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Archer

THANK YOU !!!