Databases

Data model

A **data model** is a conceptual representation to express and communicate business requirements. It visually represents the nature of data, business rules governing the data, and how the data will be organized in the database.

Key Terms

Entity and Attributes: The entities are the “things” in the business environment about which we want to store data e.g Products, Customers, Orders etc. Attributes provide a means of organizing and structuring data.

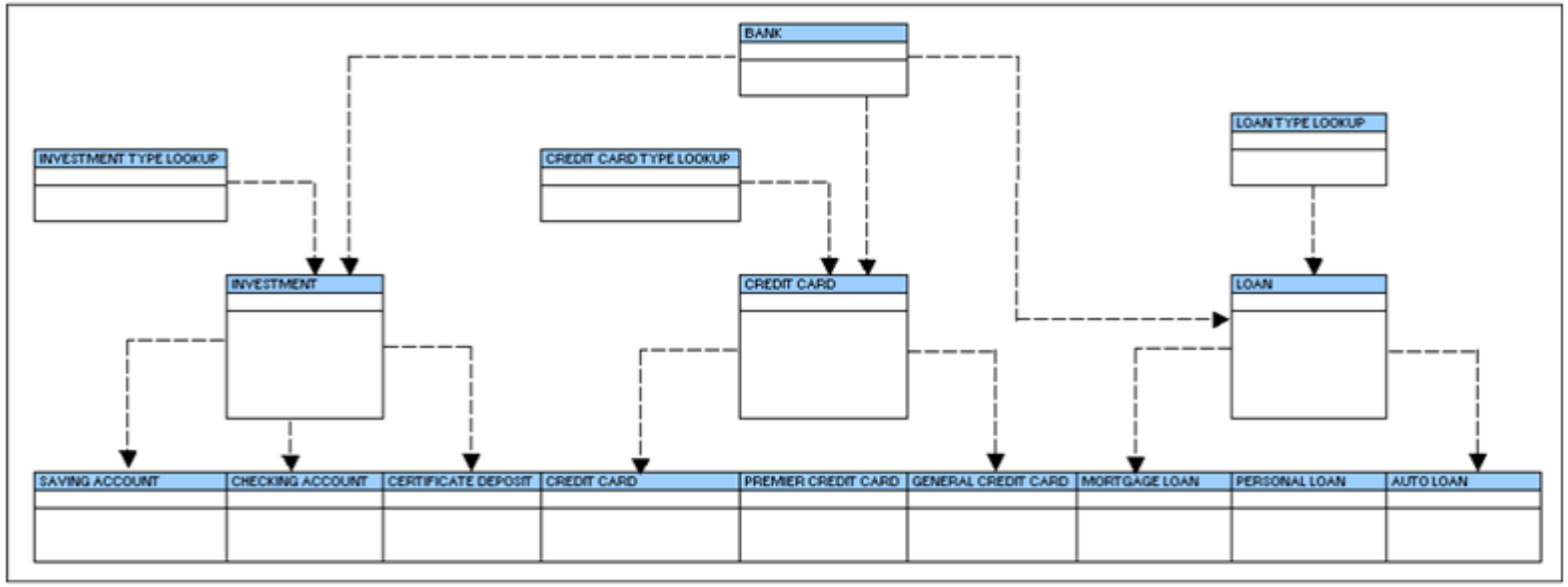
Relationship: The relationship between entities describes how one entity is linked to another. In a data model, entities can be related as any of: one-to-one, many-to-one or many-to-many. This is said to be the cardinality of a given entity in relation to another.

Intersection Entity(Reference Table): In case of many-to-many relationship among entities, an intersection entity can be used to resolve it to many to one and one to many relationships.

ER Diagram: A diagram that shows the entities and the relationships between them is called ER diagram. An ER diagram can take the form of Conceptual Data Model, Logical Data Model or Physical Data Model.

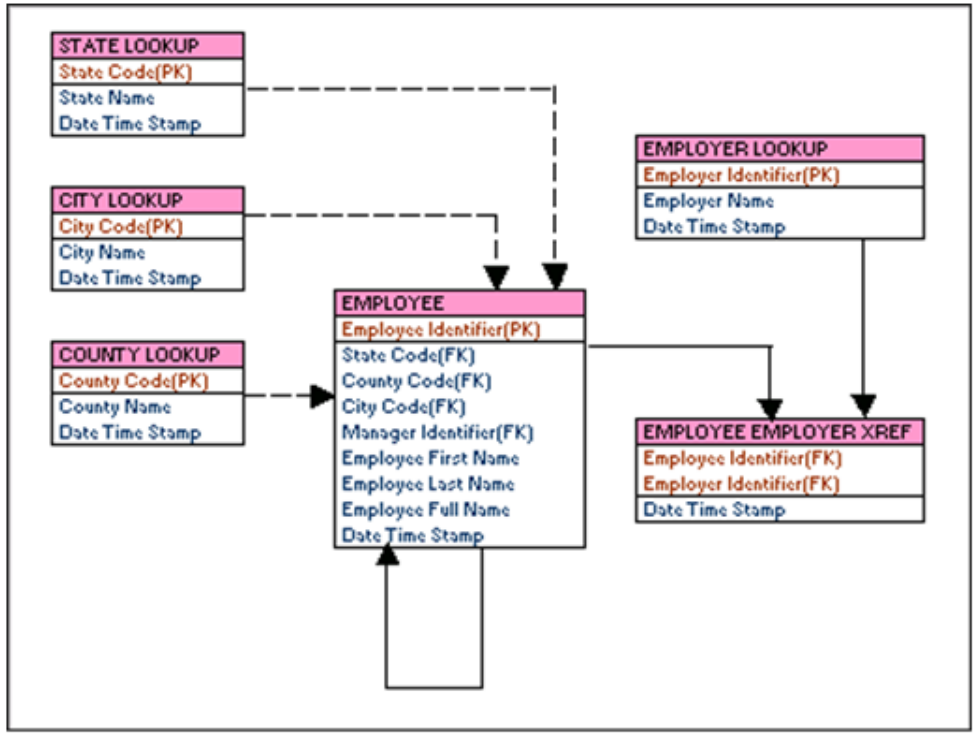
Conceptual Data Model

Conceptual data model includes all major entities and relationships and does not contain much detailed level of information about attributes and is often used in the initial planning phase.



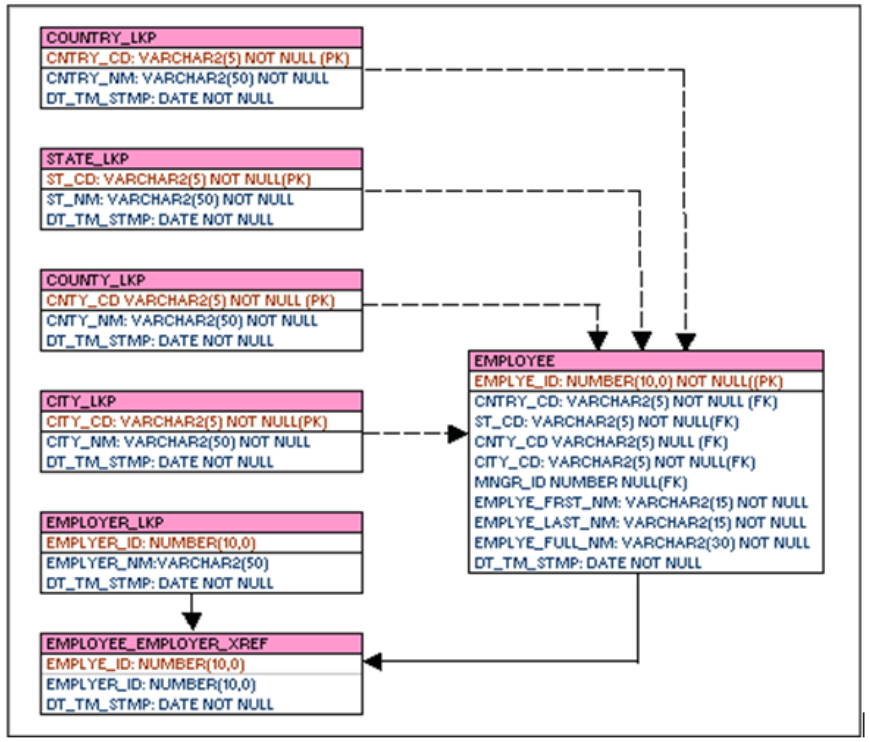
Logical Data Model

It is an extension of the conceptual data model. It includes all entities, attributes, key groups and relationships that represent business information and define business rules.



Physical Data Model

It includes all required tables, columns, relationships, database properties, for the physical implementation of databases. Database performance, indexing strategy, physical storage and denormalization are important parameters of a physical model.



Tables and attributes

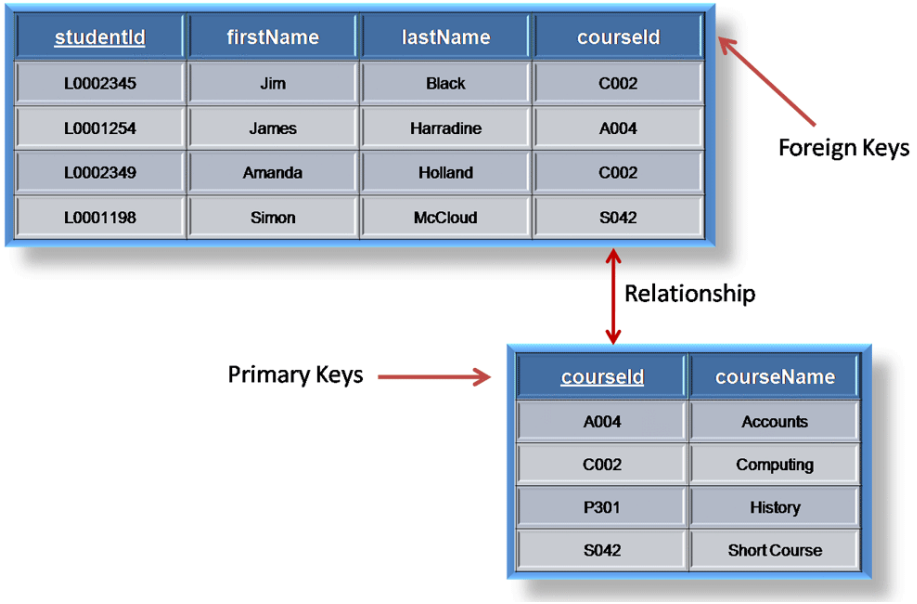
Within a relational database, records are stored in tables (think Excel spreadsheets, if that's something you're familiar with) where each column is an *attribute* (e.g product name, purchase cost, retail price) and each row represents a *record* (a particular item or instance which has those attributes). Each column has its own type.

product_id	brand	product_name	product_type	unit_cost	unit_sales_price
101	Cool Brand™	Luft Extreme	sneakers	€ 40	€ 90
101	Cool Brand™	Luft Ultra	sneakers	€ 40	€ 90
102	Rival Brand™	Aero Maxima	sneakers	€ 25	€ 75
103	Third Brand™	Ultimum Enduro	shorts	€ 7	€ 30

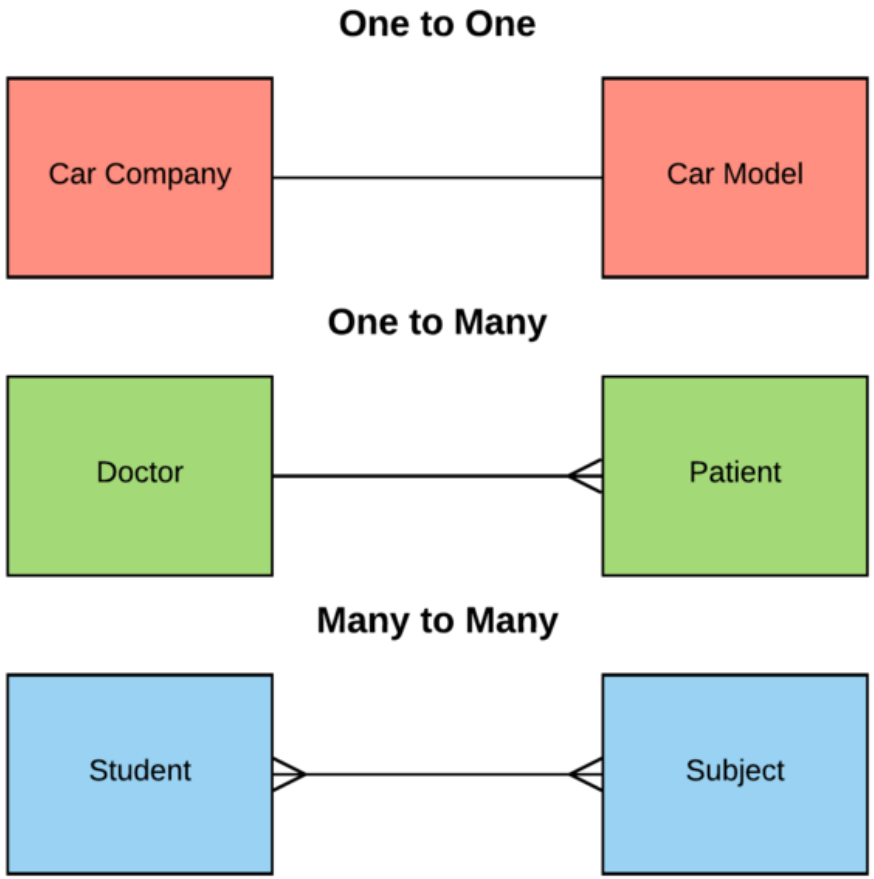
Primary and Foreign Keys

Every record in every table in your database has to have an attribute (or combination of attributes) which identifies it uniquely — this is known as the primary key.

We can express the relationship between entities by including the primary key from the one table as a ‘foreign key’ in the other.



Types of relationships

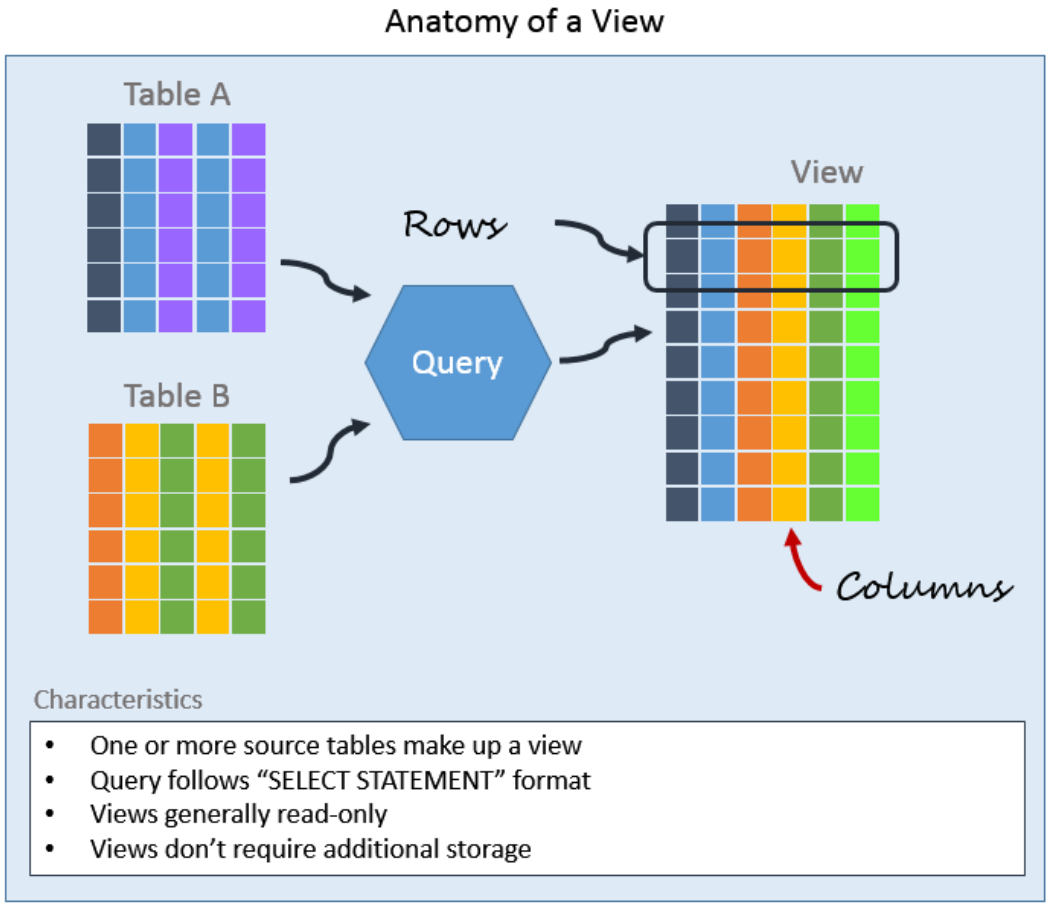


Views

A view is a logical table based on a table or another view.

A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables.

The view is stored as a SELECT statement in the data dictionary.

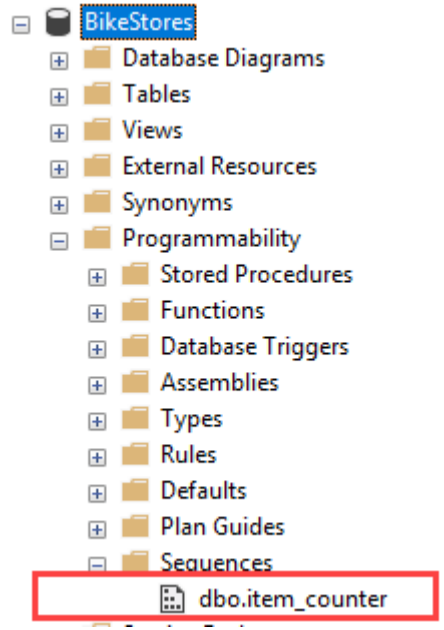


Sequence

A sequence is a user created database object that can be shared by multiple users to generate unique integers.

A typical usage for sequences is to create a primary key value, which must be unique for each row.

Often the sequence is generated and incremented (or decremented) by an internal routine.

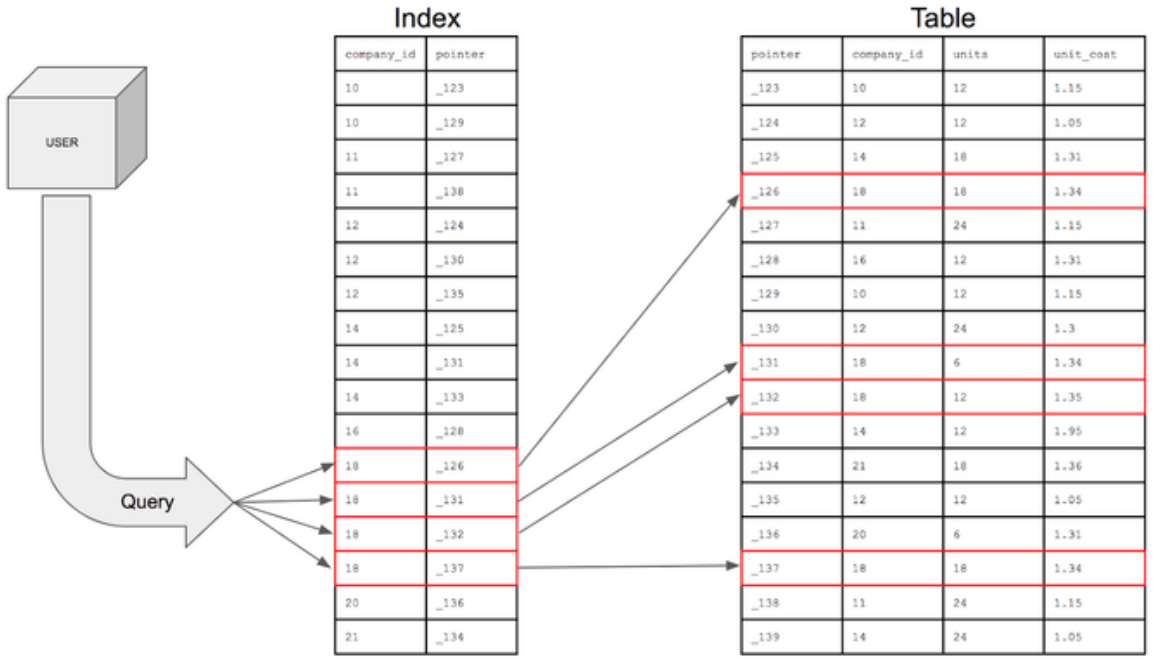


Index

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.

Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

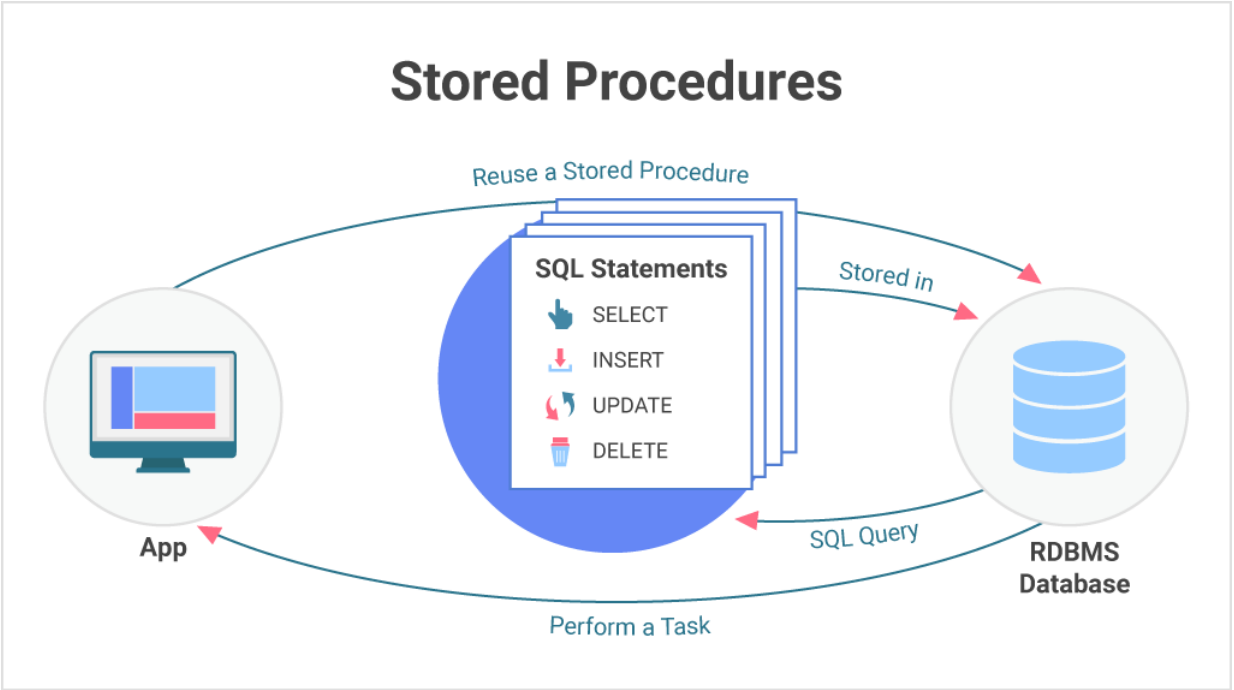


Stored functions and procedures

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.



Other objects

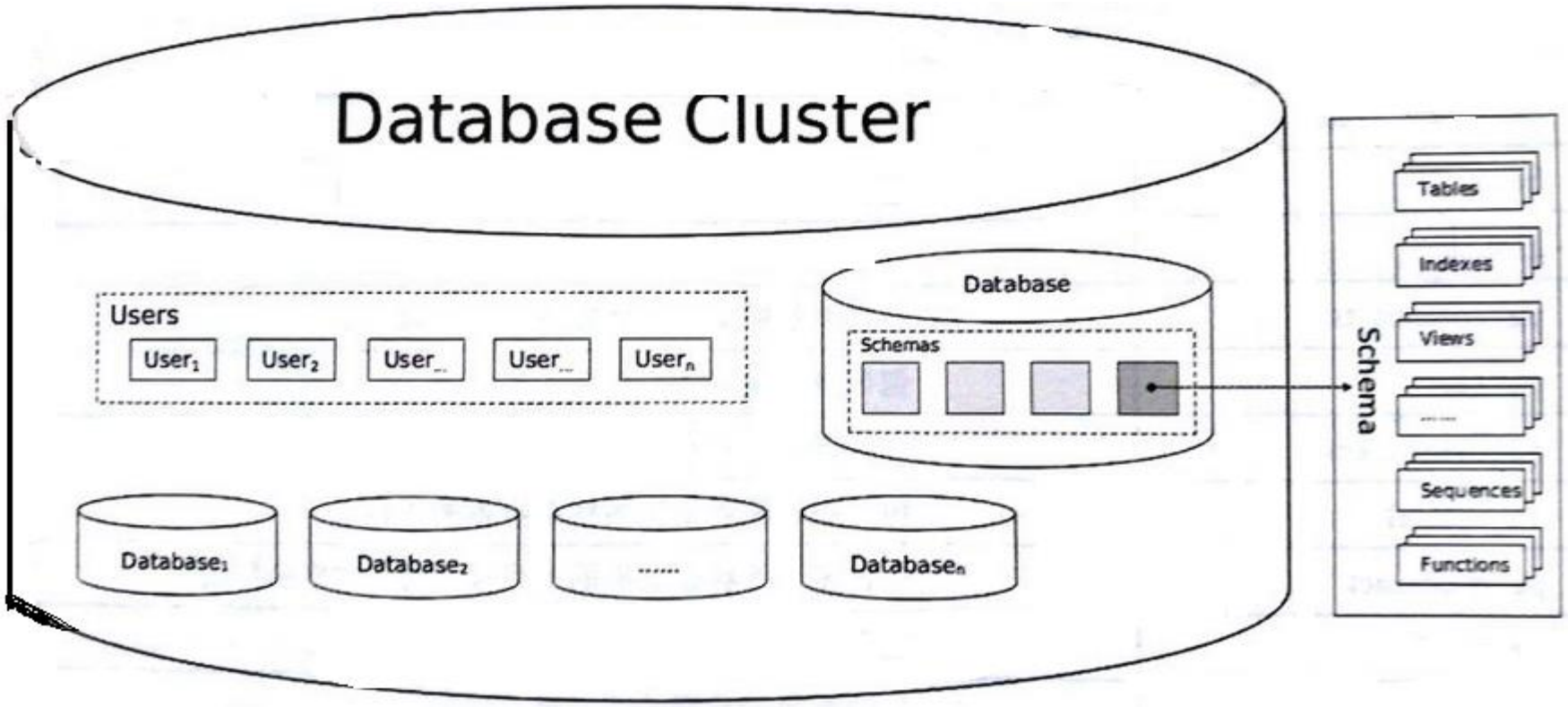
Synonyms

Users

Roles

Tablespaces

Schemas

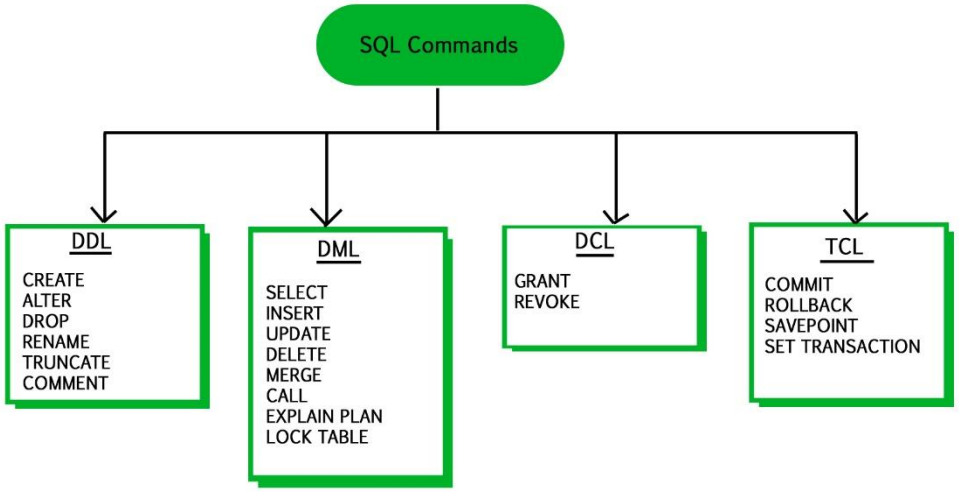


Structured Query Language

Domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS). Originally based upon relational algebra and tuple relational calculus.

SQL consists of many types of statements, which may be informally classed as sublanguages, commonly:

- data definition language (DDL),
- data control language (DCL),
- data manipulation language (DML),
- transaction control language (TCL)



DDL

CREATE statement

DROP statement

ALTER statement

TRUNCATE statement

Constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Constraints

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

DML

SELECT: retrieve data from the database

INSERT: insert data into a table

UPDATE: update existing data within a table

DELETE: deletes all records from a table, space for the records remain

SELECT

SELECT [DISTINCT] column1, column2, columnN [AS alias]

FROM table_name [AS tbl_alias]

[WHERE condition]

[GROUP BY column1, column2]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

JOINS

There are different types of joins available in SQL −

INNER JOIN − returns rows when there is a match in both tables.

LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.

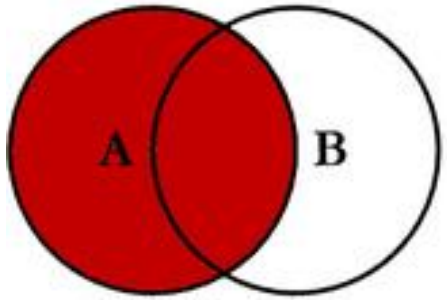
RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.

FULL JOIN − returns rows when there is a match in one of the tables.

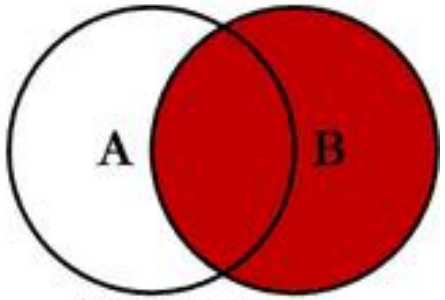
SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables.

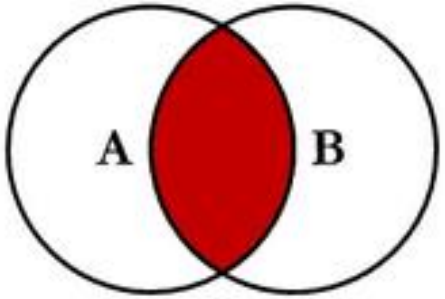
SQL JOINS



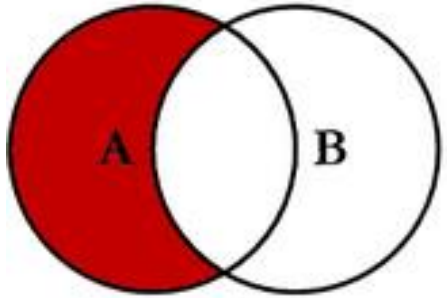

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



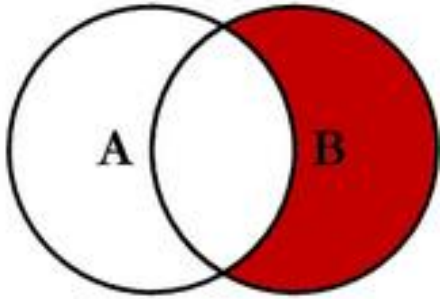

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



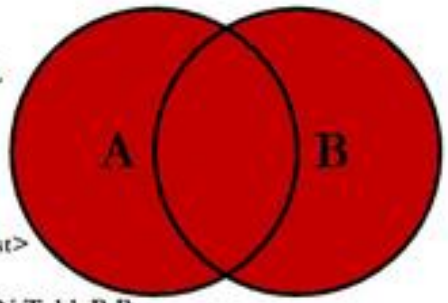

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



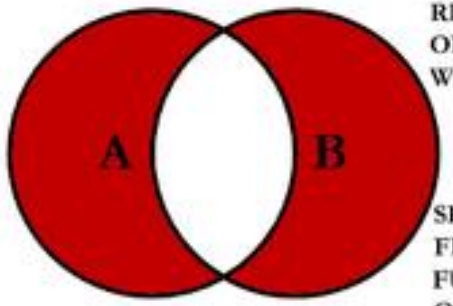

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

WITH (CTE)

The **Common Table Expressions (CTE)** were introduced into standard SQL in order to simplify various classes of SQL Queries for which a derived table was just unsuitable.

CTE was introduced in SQL Server 2005, the common table expression (CTE) is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. You can also use a CTE in a CREATE a view, as part of the view's SELECT query. In addition, as of SQL Server 2008, you can add a CTE to the new MERGE statement.

```
WITH expression_name[(column_name [,...])]

AS

   (CTE_definition)

SQL_statement;
```

WITH (Recursive CTE)

```
CREATE TABLE Employees
(
  EmployeeID int NOT NULL PRIMARY KEY,
  FirstName varchar(50) NOT NULL,
  LastName varchar(50) NOT NULL,
  ManagerID int NULL
)

INSERT INTO Employees VALUES (1, 'Ken', 'Thompson', NULL)
INSERT INTO Employees VALUES (2, 'Terri', 'Ryan', 1)
INSERT INTO Employees VALUES (3, 'Robert', 'Durello', 1)
INSERT INTO Employees VALUES (4, 'Rob', 'Bailey', 2)
INSERT INTO Employees VALUES (5, 'Kent', 'Erickson', 2)
INSERT INTO Employees VALUES (6, 'Bill', 'Goldberg', 3)
INSERT INTO Employees VALUES (7, 'Ryan', 'Miller', 3)
INSERT INTO Employees VALUES (8, 'Dane', 'Mark', 5)
INSERT INTO Employees VALUES (9, 'Charles', 'Matthew', 6)
INSERT INTO Employees VALUES (10, 'Michael', 'Jhonson', 6)
```

```
WITH
  cteReports (EmpID, FirstName, LastName, MgrID, EmpLevel)
  AS
  (
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID,
      r.EmpLevel + 1
    FROM Employees e
      INNER JOIN cteReports r
        ON e.ManagerID = r.EmpID
  )
SELECT
  FirstName + ' ' + LastName AS FullName,
  EmpLevel,
  (SELECT FirstName + ' ' + LastName FROM Employees
    WHERE EmployeeID = cteReports.MgrID) AS Manager
FROM cteReports
ORDER BY EmpLevel, MgrID
```