DBMS Introduction

Unit - 1

Course Code	18CSC303J	Course Name	DATABASE MANAGEMENT SYSTEMS	Course Category	C	Professional Core	L	T	P	C
							3	0	2	4

Pre-requisite Courses	Nil	Co-requisite Courses	Nil	Progressive Courses	Nil
Course Offering Department	Computer Science and Engineering	Data Book / Codes/Standards	Nil		

Course Learning Rationale (CLR):	The purpose of learning this course is to:	Learning			Program Learning Outcomes (PLO)														
		1	2	3	1	2	3	4	5	6	7	8	9	100	11	12	13	14	15
		Level of Thinking (Bloom)	Expected Proficiency (%)	Expected Attainment (%)	Engineering Knowledge	Problem Analysis	Design & Development	Analysis, Design, Research	Modern Tool Usage	Society & Culture	Environment & Sustainability	Ethics	Individual & Team Work	Communication	Project Mgt. & Finance	Life Long Learning	PSO - 1	PSO - 2	PSO – 3
CLR-1 :	Understand the fundamentals of Database Management Systems, Architecture and Languages																		
CLR-2 :	Conceive the database design process through ER Model and Relational Model																		
CLR-3 :	Design Logical Database Schema and mapping it to implementation level schema through Database Language																		
CLR-4 :	Familiarize queries using Structure Query Language (SQL) and PL/SQL																		
CLR-5 :	Familiarize the Improvement of the database design using normalization criteria and optimize queries																		
CLR-6 :	Understand the practical problems of concurrency control and gain knowledge about failures and recovery																		
Course Learning Outcomes (CLO):	At the end of this course, learners will be able to:																		
CLO-1 :	Acquire the knowledge on DBMS Architecture and Languages	3	80	70	H	M	L	L	-	-	-	-	L	L	L	H	-	-	-
CLO-2 :	Apply the fundamentals of data models to model an application's data requirements using conceptual modeling tools like ER diagrams	3	85	75	H	H	H	H	H	-	-	-	H	H	H	H	-	-	-
CLO-3 :	Apply the method to convert the ER model to a database schemas based on the conceptual relational model	3	75	70	H	H	H	H	H	-	-	-	H	H	H	H	-	-	-
CLO-4 :	Apply the knowledge to create, store and retrieve data using Structure Query Language (SQL) and PL/SQL	3	85	80	H	H	H	H	H	-	-	-	H	H	H	H	-	-	-
CLO-5 :	Apply the knowledge to improve database design using various normalization criteria and optimize queries	3	85	75	H	H	L	M	L	-	-	-	M	M	M	L	-	-	-
CLO-6 :	Appreciate the fundamental concepts of transaction processing- concurrency control techniques and recovery procedures.	3	85	75	H	L	L	L	L	-	-	-	H	L	L	L			

Duration (hour)		15	15	15	15	15
S-1	SLO-1	What is Database Management System	Database Design	Basics of SQL-DDL,DML,DCL,TCL	Relational Algebra – Fundamental Operators and syntax, relational algebra queries, Tuple relational calculus	Transaction concepts, properties of transactions,
	SLO-2	Advantage of DBMS over File Processing System	Design process	Structure Creation, alternation		
S-2	SLO-1	Introduction and applications of DBMS	Entity Relation Model	Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator		Serial izability of transactions,
	SLO-2	Purpose of database system				testing for serial inability, System recovery,
S-3	SLO-1	Views of data	ER diagram	Functions-aggregation functions	Pitfalls in Relational database, Decomposing bad schema	Concurrency Control
	SLO-2			Built-in Functions-numeric, date, string functions, string functions, Set operations,	Functional Dependency – definition, trivial and non-trivial FD	
S 4-5	SLO-1	Lab 1: SQL Data Definition Language Commands on sample exercise * The abstract of the project to construct database must be framed	Lab4 Inbuilt functions in SQL on sample Exercise.	Lab 7 : Join Queries on sample exercise. * Frame and execute the appropriate DDL,DML,DCL,TCL for the project	Lab10: PL/SQL Procedures on sample exercise. * Frame and execute the appropriate Join Queries for the project	Lab 13: PL/SQL Exception Handling * Frame and execute the appropriate PL/SQL Procedures and Functions for the project
	SLO-2					
S-6	SLO-1	Database system Architecture	Keys , Attributes and Constraints	Sub Queries, correlated sub queries	closure of FD set , closure of attributes	Two- Phase Commit protocol, Recovery and Atomicity
	SLO-2				irreducible set of FD	
S-7	SLO-1	Data Independence	Mapping Cardinality	Nested Queries, Views and its Types	Normalization – 1Nf, 2NF, 3NF,	Log-based recovery
	SLO-2					
S-8	SLO-1	The evolution of Data Models	Extended ER - Generalization,	Transaction Control Commands	Decomposition using FD- dependency	concurrent executions of transactions and
	SLO-2		Specialization and Aggregation	Commit, Rollback, Save point	preservation,	related problems
S 9-10	SLO-1	Lab 2: SQL Data Manipulation	Lab 5: Construct a ER Model for the	Lab 8: Set Operators & Views.	Lab 11: PL/SQL Functions	Lab 14: PL/SQL Trigger

	SLO-2	*Language Commands * Identification of project Modules and functionality*	*application to be constructed to a Database*	** Frame and execute the appropriate In- Built functions for the project*	** Frame and execute the appropriate Set Operators & Views for the project*	** Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project*
S-11	SLO-1	*Degrees of Data Abstraction*	*ER Diagram Issues*	*PL/SQL Concepts- Cursors*	*BCNF*	*Locking mechanism, solution to concurrency related problems*
	SLO-2		*Weak Entity*			
S-12	SLO-1	*Database Users and DBA*	*Relational Model*	*Stored Procedure, Functions Triggers and Exceptional Handling*	*Multi- valued dependency,*	*Deadlock*
	SLO-2				*4NF*	
S-13	SLO-1	*Database Languages*	*Conversion of ER to Relational Table*	*Query Processing*	*Join dependency and 5NF*	*two-phase locking protocol, Isolation, Intent locking*
	SLO-2					
S 14-15	SLO-1	*Lab 3: SQL Data Control Language Commands and Transaction control commands to the sample exercises*	*Lab 6: Nested Queries on sample exercise*	*Lab9: PL/SQL Conditional and Iterative Statements*	*Lab 12: PL/SQL Cursors*	*Lab 15 : * Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project*
	SLO-2	** Identify the issues that can arise in a business perspective for the application*	** Construction of Relational Table from the ER Diagram*	** Frame and execute the appropriate Nested Queries for the project*	** Frame and execute the appropriate PL/SQL Conditional and Iterative Statements for the project*	** Demo of the project*

Learning Resources	1. *Abraham Silberschatz, Henry F. Korth, S. Sudharshan, Database System Concepts‖, Sixth Edition, Tata McGraw Hill,2011.* 2. *Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Sysiems‖, Sixth Edition, Pearson Education,2011.* 3. *CJ Date,A Kannan,S Swamynathan, An Introduction to Database Systems, Eight Edition, Pearson Education,2006.* 4. *Rajesh Narang, Database Management Systems, 2nd ed., PHI Learning Private Limited,2011.*	4. *Martin Gruber, Understanding SQL, Sybex,1990* 5. *Sharad Maheshwari,Introduction to SQLandPL/SQL,2nded.,LaxmiPublications,2016.* 6. *RaghuramaKrishnan,Johannes Gehrke,Database Management Systems,3rdEdition,McGrawHill Education,2003.*

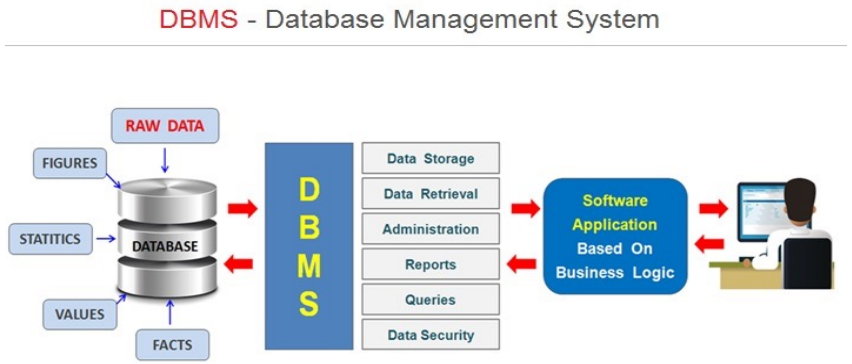
Learning Assessment

	Bloom's Level of Thinking	Continuous Learning Assessment (50% weightage)								Final Examination (50% weightage)	
		CLA – 1 (10%)		CLA – 2 (15%)		CLA – 3 (15%)		CLA – 4 (10%)#			
		Theory	Practice	Theory	Practice	Theory	Practice	Theory	Practice	Theory	Practice
Level 1	Remember	*20%*	*20%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*
	Understand										
Level 2	Apply	*20%*	*20%*	*20%*	*20%*	*20%*	*20%*	*20%*	*20%*	*20%*	*20%*
	Analyze										
Level 3	Evaluate	*10%*	*10%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*	*15%*
	Create										
	Total	100 %		100 %		100 %		100 %		-	

CLA – 4 can be from any combination of these: Assignments, Seminars, Tech Talks, Mini-Projects, Case-Studies, Self-Study, Conf. Paper etc.,

Course Designers		
Experts from Industry	Experts from Higher Technical Institutions	Internal Experts
1. Dr.Mariappan Vaithilingam, Engineering Leader Amazon, dr.v.m@ieee.org		*1. Ms. Sasi Rekha Sankar SRMIST*
2. Mr. Badinath, SDET, Amzon, sbadhrinath@gmail.com		*2. Mr.Elizer, SRMIST*
		3. Mrs. Hemavathy, SRMIST

DBMS – Data Base Management System



- A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data.
- Database - The collection of data, that contains information relevant to an enterprise.
- DBMS is to provide a way to store and retrieve database information.

Importance

- Database systems are designed to manage large bodies of information.
- Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.
- Ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.
- If data are to be shared among several users, the system must avoid possible anomalous results.

Database-System Applications

- **Banking:** all transactions.
- **Airlines:** reservations, schedules.
- **Universities:** registration, grades.
- **Sales:** customers, products, purchases.
- **Manufacturing:** production, inventory, orders, supply chain.
- **Human resources:** employee records, salaries, tax deductions.
- **Credit card transactions:** Purchase, generation of monthly statements.
- **Finance:** storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
- **Telecommunication:** keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Purpose of Database Systems

File-processing system

- Supported by a conventional operating system.
- The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.
- Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems.

Disadvantage

Data redundancy: Multiple file formats, duplication of information in different files.

- **E.g.:** Student has a double major (music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department.
- This redundancy leads to **higher storage** and **access cost**.

Data inconsistency: The various copies of the same data may no longer agree.

- **E.g.:** A changed student address may be reflected in the Music department records but not elsewhere in the system.

Difficulty in accessing data: Need to write a new program to carry out each new task.

- Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- More responsive data-retrieval systems are required for general use.

Data isolation: Data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Integrity problems: The data values stored in the database must satisfy certain types of consistency constraints.

- **E.g.:** account balance < 0 (never fall less than zero).
- Hard to add new constraints or change existing ones.

Atomicity problems: Failures may leave the database in an inconsistent state with partial updates carried out.

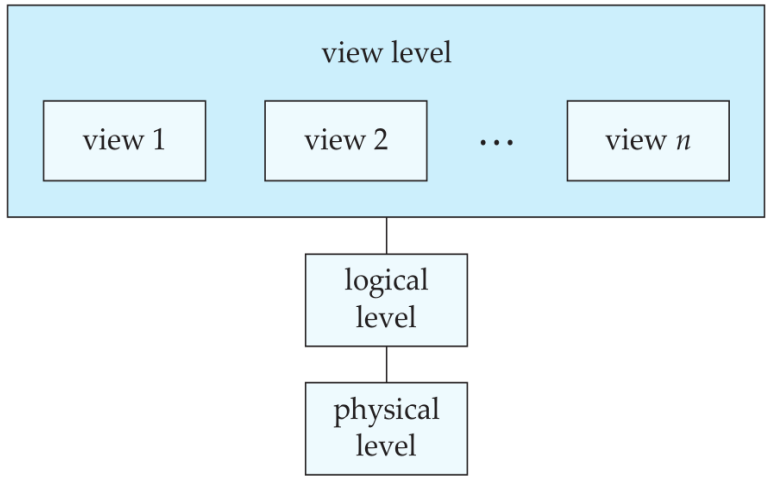
- **E.g.:** Transfer of funds from one account to another should either be complete or not happen at all.
- Difficult to ensure atomicity in a conventional file-processing system.

- **Concurrent-access anomalies:** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
- Uncontrolled concurrent accesses can lead to inconsistencies.
- **E.g.:** Two people reading a balance and updating it at the same time.
- Consider department A, with an account balance of $10,000. If two department clerks debit the account balance ($500 and $100) of department A at almost exactly the same time the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state.
- If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively. Depending on which one writes the value last, the account balance of department A may contain either $9500 or $9900, rather than the correct value of $9400.

- **Security problems:** Not every user of the database system should be able to access all the data.
- **E.g.:** in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records.

Database systems offer solutions to all the above problems

Levels of Data Abstraction



- **Physical level:** The lowest level of abstraction.
- Describes how the data are actually stored.
- The physical level describes complex low-level data structures in detail.
- **Logical level:** Describes what data are stored in the database, and what relationships exist among those data.

```
type customer = record
                    name : string;
                    street : string;
                    city : integer;
                end;
```

- **View level:** The highest level of abstraction. Describes only part of the entire database. Application programs hide details of data types.
- Views can also hide information (e.g., salary) for security purposes.

- CREATE VIEW [Brazil Customers] AS
  SELECT CustomerName, ContactName
  FROM Customers
  WHERE Country = 'Brazil';

- CREATE VIEW V AS
- (SELECT * FROM POP
- UNION
- SELECT * FROM FOOD
- UNION
- SELECT * FROM INCOME);

Instances and Schemas

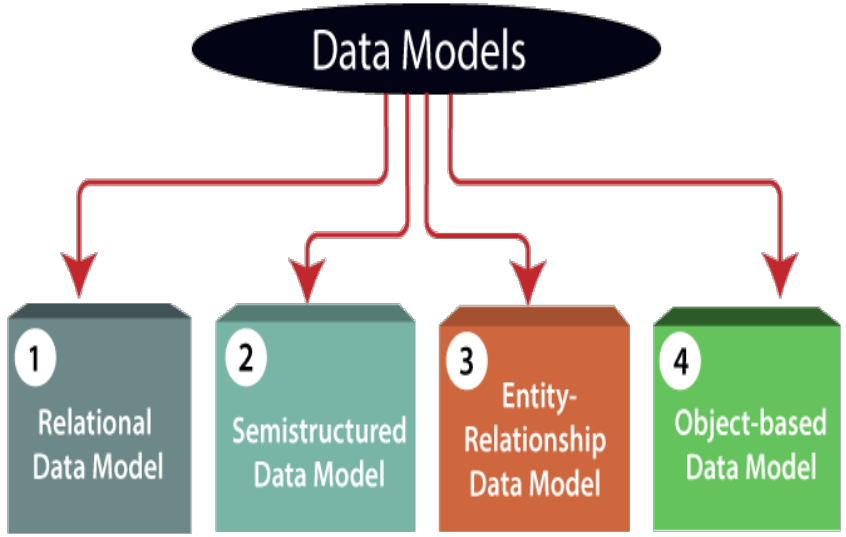
- **Instance:** The collection of information stored in the database at a particular moment.
- **Schema:** The overall design of the database is called the database schema.
- The concept of database schemas and instances can be understood by analogy to a program written in a programming language.

- A **database schema** corresponds to the **variable declarations** (along with associated type definitions) in a program.
- The **values of the variables** in a program at a point in time correspond to an **instance** of a database schema.

- **Physical schema:** Describes the database design at the physical level.
- **Logical schema:** Describes the database design at the logical level.
- **Subschemas:** A database may also have several schemas at the view level.

- **Physical Data Independence:** The ability to modify the physical schema without changing the logical schema.
  - Applications depend on the logical schema.
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Data Models



- A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- A data model provides a way to describe the design of a database at the physical, logical, and view levels.

- **Relational Model:** The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name.
- Tables are also known as relations. The relational model is an example of a record-based model.

- **Entity-Relationship Model:** The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects.
- The entity-relationship model is widely used in database design.

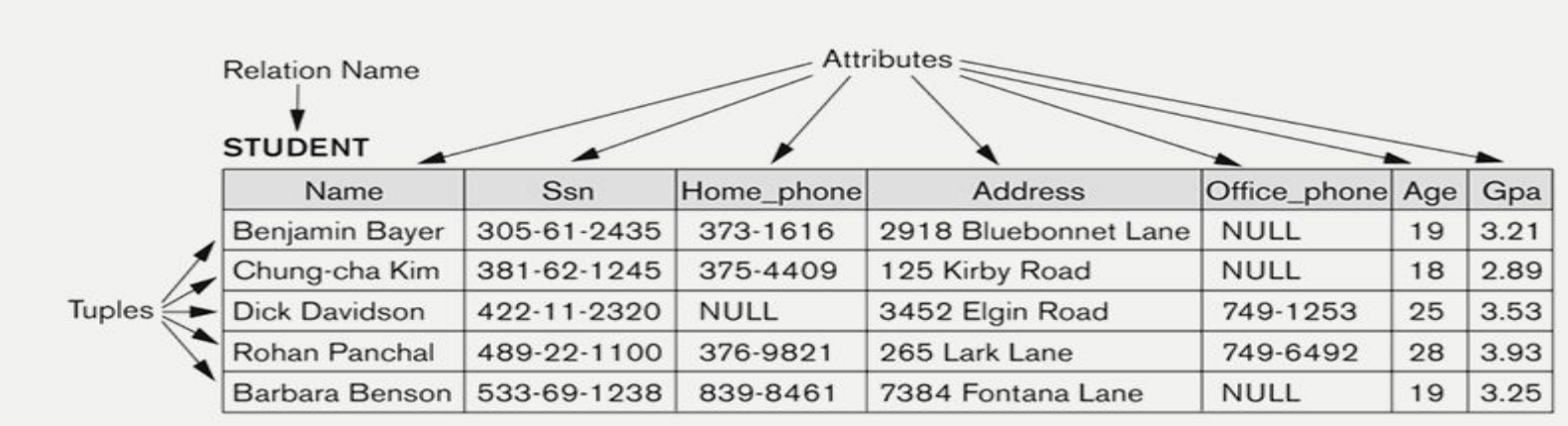
- **Object-Based Data Model:** The development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.
- The object-relational data model combines features of the object-oriented data model and relational data model.

- **Semi structured Data Model:** The semi structured data model permits the specification of data where individual data items of the same type may have different sets of attributes.
- The Extensible Markup Language (XML) is widely used to represent semi structured data.

The network data model and the hierarchical data model preceded the relational data model.

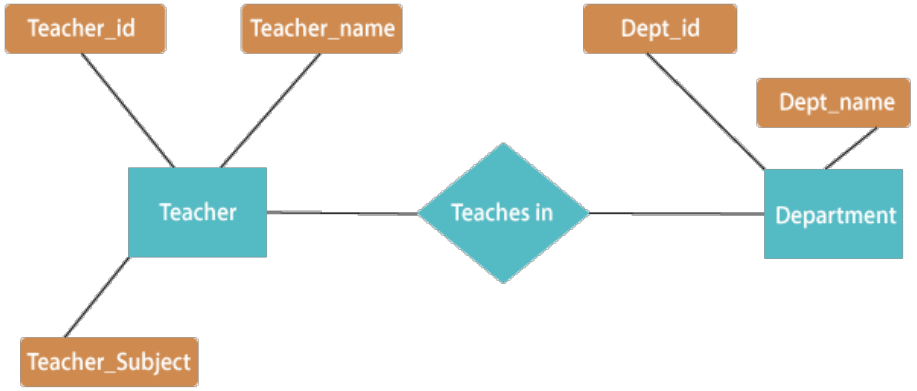
Relational Model

- Tuple – one row of a relation
- Attribute – one column of a relation
- Relation – the whole table
- Domain of an attribute – all the values that the attribute



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

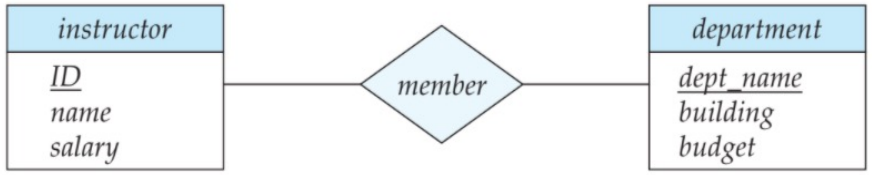
Entity-Relationship Model



- E-R model of real world
  - Entities (objects)
    - E.g. Teacher, Department.
  - Relationships between entities
    - Relationship set *teaches in* associates teacher with department.

- Widely used for database design.

- **Entity sets** are represented by a rectangular box with the entity set name in the header and the attributes listed below it.

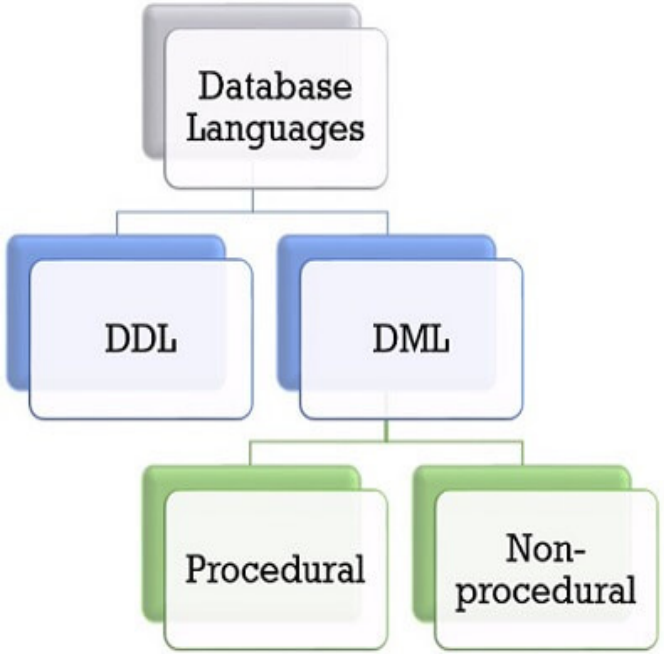


- **Relationship sets** are represented by a diamond connecting a pair of related entity sets.
- The name of the relationship is placed inside the diamond.

- **Mapping cardinalities:** The number of entities to which another entity can be associated via a relationship set.

Database Languages

- A database system provides a **Data-Definition Language (DDL)** to specify the database schema and a **Data-Manipulation Language (DML)** to express database queries and updates.



Data-Manipulation Language (DML)

- A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.

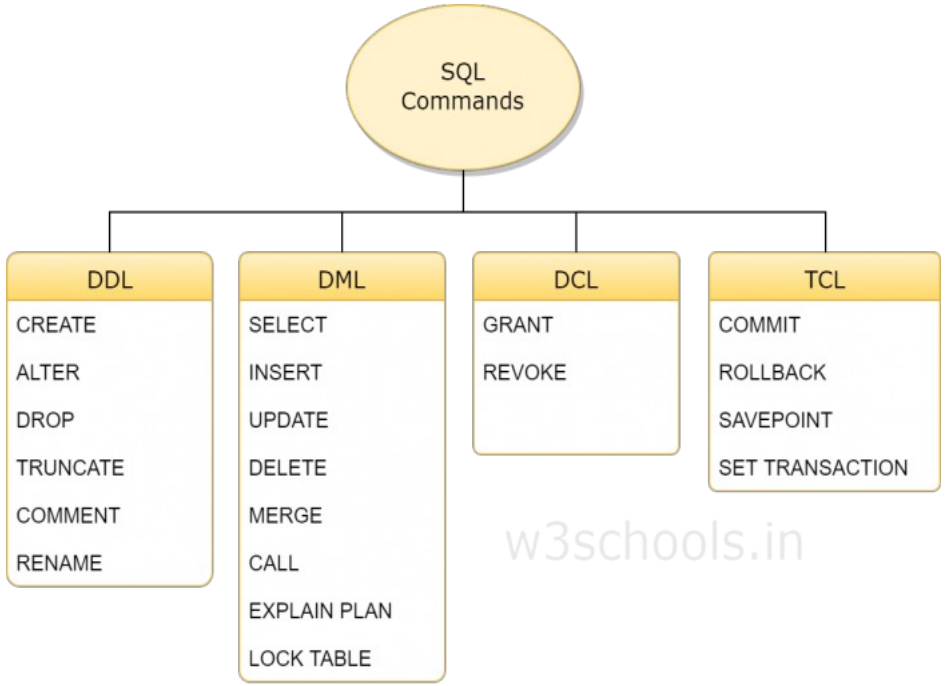
The types of access are

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

- Procedural DMLs require a user to specify what data are needed and how to get those data.
- Declarative DMLs (also called as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

- A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language.

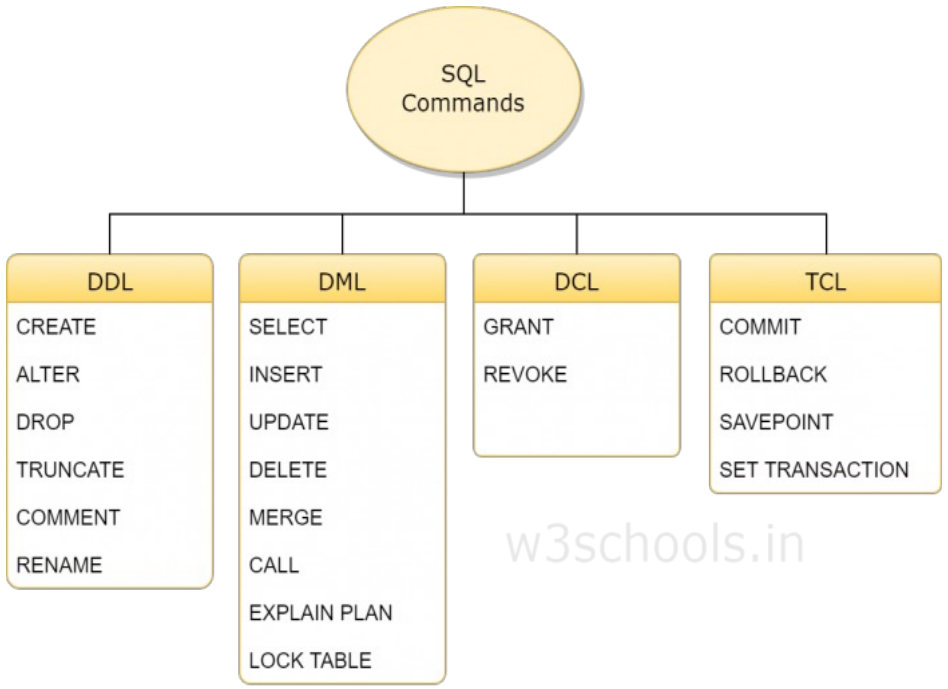
Database Languages



DDL is Data Definition Language statements

- Some examples

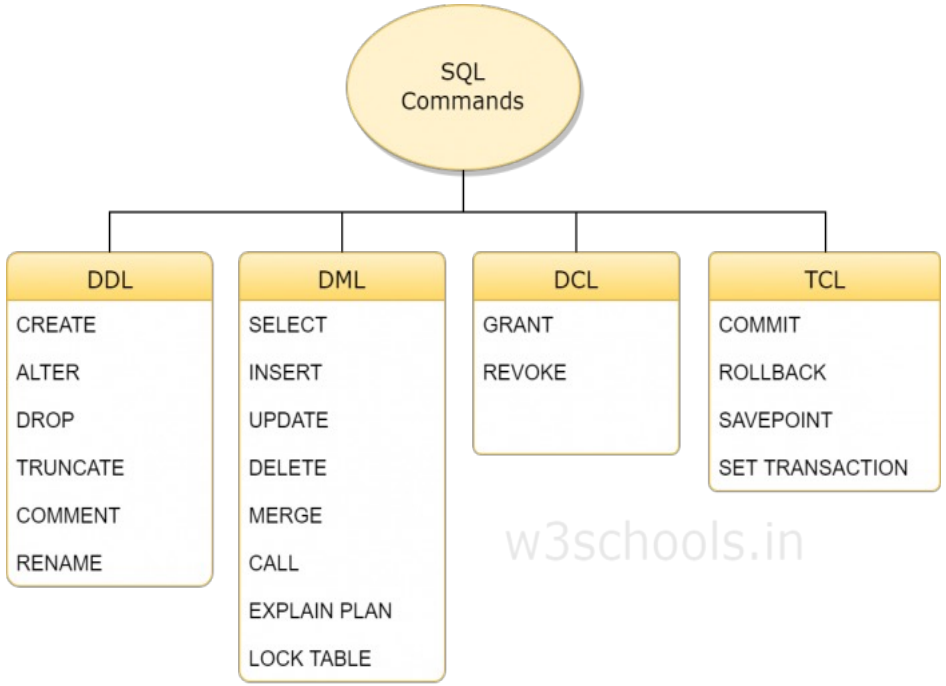
- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table
- COMMENT - add comments to the data dictionary



DML is Data Manipulation Language statements

- Some examples:

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency



TCL(Transaction Control Language) is a DML

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like what rollback segment to use

DCL is Data Control Language statements

- Some examples:
- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command
- GRANT SELECT ON Users TO'Amit'@'localhost;

Data-Definition Language

- Set of definitions expressed by a special language called a data-definition language (DDL).
- The storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language.
- The data values stored in the database must satisfy certain consistency constraints.
- **Domain Constraints:** A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
- Domain constraints are the most elementary form of integrity constraint.
- **Referential Integrity:** There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).
- Database modifications can cause violations of referential integrity.
- **Assertions:** An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions.
- **Authorization:** To differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of authorization.

- Read Authorization - which allows reading, but not modification of data.
- Insert Authorization - which allows insertion of new data but not modification of existing data.
- update authorization - which allows modification but not deletion of data.
- Delete Authorization - which allows deletion of data.

- The output of the DDL is placed in the data dictionary which contains metadata - that is, data about data.
- SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc.

create table department (dept name char (20) primary key, building char (15), budget numeric (12,2));

- Execution of the above DDL statement creates the department table with three columns: dept name, building, and budget, each of which has a specific data type associated with it.

CREATING DATABASE TABLE

- CREATE – creates a new table in the database

- Used to create a table by defining its structure, the data type and name of the various columns, the relationships with columns of other tables etc.

- CREATE TABLE table_name (column_name1 data_type(size), column_name2 data_type(size),…., column_nameN data_type(size));

- E.g.:

  CREATE TABLE Employee(Name varchar2(20), DOB date, Salary number(6));

ALTER - Add a new attribute or Modify the characteristics of some existing attribute.

- ALTER TABLE table_name ADD (column_name1 data_type (size), column_name2 data_type (size),….., column_nameN data_type (size));

E.g.:

ALTER TABLE Employee ADD (Address varchar2(20));

ALTER TABLE Employee ADD (Designation varchar2(20), Dept varchar2(3));

ALTER TABLE table_name MODIFY (column_name data_type(new_size));

E.g.:

ALTER TABLE Employee MODIFY (Name varchar2(30));

ALTER - dropping a column from the table

- ALTER TABLE table_name DROP COLUMN column_name;

E.g.:

ALTER TABLE Student DROP COLUMN Age;

DROP - Deleting an entire table from the database.

DROP TABLE table_name;

E.g.:

DROP TABLE Employee

RENAME – Renaming the table

RENAME old_table_name TO new_table_name;

E.g.:

RENAME Employee TO Employee_details

- TRUNCATE – deleting all rows from a table and free the space containing the table.

TRUNCATE TABLE table_name;

E.g.:

TRUNCATE TABLE Employee_details;

Data Manipulation Language

A DML statement is executed when you

- Add new rows to a table
- Modify existing rows in a table
- Remove existing rows from a table

Add new rows to a table by using the INSERT statement.

1. INSERT INTO table VALUES(value1, value2,..);

- Only one row is inserted at a time with this syntax.
- List values in the default order of the columns in the table
- Enclose character and date values within single quotation marks.
- Insert a new row containing values for each column.

E.g.:

- INSERT INTO Employee VALUES (‘ashok’, ‘16-mar-1998’, 30000);

2. INSERT INTO table(column1, column2,..)VALUES(value1, value2,..);

- Rows can be inserted with NULL values either
  - by omitting column from the column list or
  - by specifying NULL in the value field.

E.g.:

- INSERT INTO Employee (name, dob, salary) VALUES (‘ashok’, ‘16-mar-1998’, 30000);

3. INSERT INTO table_name1 SELECT column_name1, column_name2,….,column_nameN FROM table_name2;

- INSERT INTO Employee_details SELECT name, dob FROM Employee;

Data-Manipulation Language (DML)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

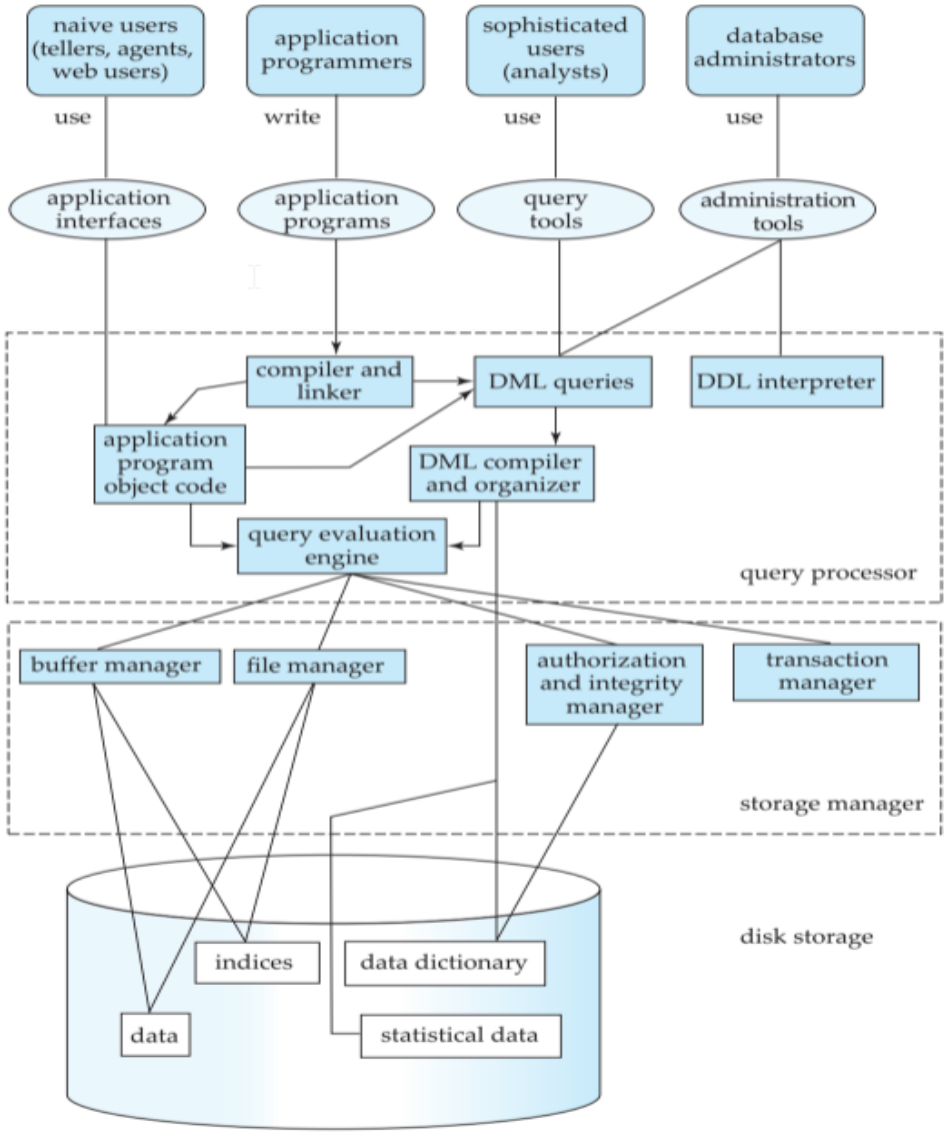
- The SQL query language is nonprocedural.

select instructor.name from instructor where instructor.dept name = 'History';

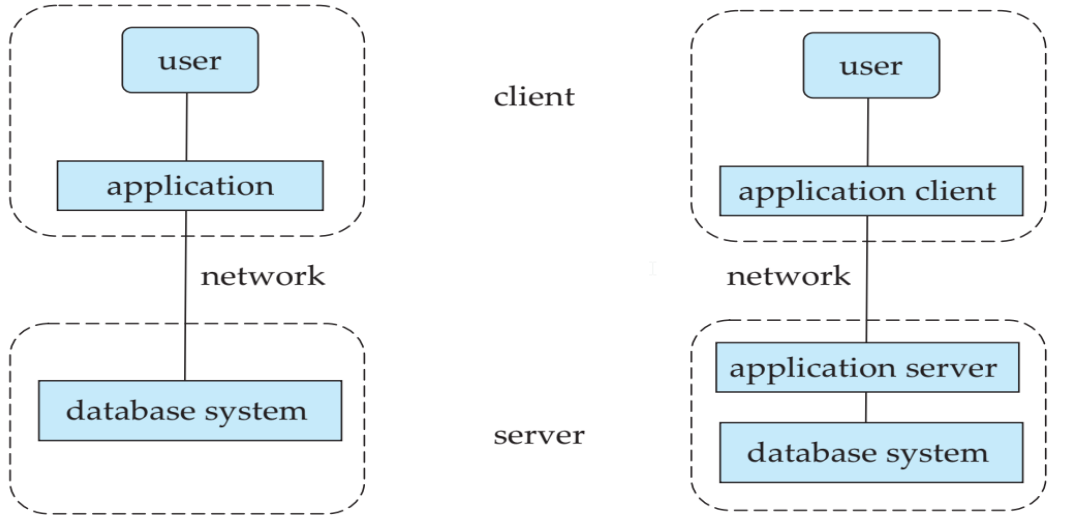
- Queries may involve information from more than one table.

select instructor.ID, department.dept name from instructor, department where instructor.dept name= department.dept name and department.budget > 95000;

Database Architecture



- Database applications are usually partitioned into two or three parts.
- In a two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements.
- Three-tier architecture - the client machine acts as merely a front end and does not contain any direct database calls.



(a) Two-tier architecture (b) Three-tier architecture

Database Users

- Users are differentiated by the way they expect to interact with the system

- **Naive users:** unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

- **Application programmers:** computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

- **Sophisticated users:** interact with the system without writing programs.

- **Specialized users:** sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Database Administrator

- One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data.
- A person who has such central control over the system is called a database administrator (DBA).
- **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.

- **Storage structure and access-method definition.**

- **Schema and physical-organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

- **Granting of authorization for data access:** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.

- **Routine maintenance:** Periodically backing up the database, Ensuring that enough free disk space, Monitoring jobs running on the database.

Transaction Management

- **Atomicity** - all-or-none requirement.
- one department account (A) is debited and another department account (B) is credited.
- Either both the credit and debit occur, or that neither occur.
- **Consistency** - it is essential that the execution of the funds transfer preserve the consistency of the database. The value of the sum of the balances of A and B must be preserved.
- This is called correctness requirement.
- **Durability** - After the successful execution of a funds transfer, the new values of the balances of accounts A and B must persist, despite the possibility of system failure.
- This is called persistence requirement.

Storage Manager

- The storage manager is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the interaction with the file manager.

The storage manager components include

- **Authorization and integrity manager:** which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
- **Transaction manager:** which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
- **File manager:** which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- **Buffer manager:** which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.
- The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

- The storage manager implements several data structures as part of the physical system implementation
- **Data files:** which store the database itself.
- **Data dictionary:** which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices:** which can provide fast access to data items.
- Like the index in this textbook, a database index provides pointers to those data items that hold a particular value.

The Query Processor

- The query processor components include
- **DDL interpreter:** which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**: which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**; that is, it picks the lowest cost evaluation plan from among the alternatives.
- **Query evaluation engine:** which executes low-level instructions generated by the DML compiler.

DML Commands

Command	Explanation
Select	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Intersection	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria, , we also include some or all **tuples** that do not **match** the **criteria**

UNION Operator

- UNION operator is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.

- Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

```
SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions] UNION SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions];
```

- There must be same number of expressions in both SELECT statements.
- Since the UNION operator by default removes all duplicate rows from the result set, providing the UNION DISTINCT modifier has no effect on the results.
- The column names from the first SELECT statement in the UNION operator are used as the column names for the result set.

```
SELECT department_id FROM departments UNION SELECT department_id FROM employees;
```

INTERSECT Operator

- The SQLite INTERSECT operator returns the intersection of 2 or more datasets. Each dataset is defined by a SELECT statement.
- If a record exists in both data sets, it will be included in the INTERSECT results.
- If a record exists in one data set and not in the other, it will be omitted from the INTERSECT results.

```
SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions] INTERSECT SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions];
```

- There must be same number of expressions in both SELECT statements.
- The corresponding expressions must have the same data type in the SELECT statements. For example: expression1 must be the same data type in both the first and second SELECT statement.

Example - With Single Expression

```
SELECT department_id FROM departments INTERSECT SELECT department_id FROM employees;
```

WHERE conditions to the INTERSECT query

```
SELECT department_id FROM departments WHERE department_id >= 25 INTERSECT SELECT department_id FROM employees WHERE last_name = 'Anderson';
```

- **Example - With Multiple Expressions**

```
SELECT contact_id, last_name, first_name FROM contacts WHERE contact_id > 50 INTERSECT SELECT
customer_id, last_name, first_name FROM customers WHERE last_name <> 'Peterson';
```

- **ORDER BY** clause is used to sort the data in an ascending or descending order, based on one or more columns.

```
SELECT column-list FROM table_name [WHERE condition] [ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

```
SELECT * FROM COMPANY ORDER BY SALARY ASC;
```

ID	NAME	AGE	ADDRESS	SALARY
7	James	24	Houston	10000.0
2	Allen	25	Texas	15000.0
1	Paul	32	California	20000.0
3	Teddy	23	Norway	20000.0
6	Kim	22	South-Hall	45000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0

```
SELECT * FROM COMPANY ORDER BY NAME DESC
```

ID	NAME	AGE	ADDRESS	SALARY
3	Teddy	23	Norway	20000.0
1	Paul	32	California	20000.0
4	Mark	25	Rich-Mond	65000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0
5	David	27	Texas	85000.0
2	Allen	25	Texas	15000.0

- **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
SELECT column-list FROM table_name WHERE [ conditions ] GROUP BY column1, column2....columnN ORDER BY column1, column2....columnN
```

ID	NAME	AGE	ADDRESS	SALARY
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME;
```

NAME	SUM(SALARY)
Allen	15000.0
David	85000.0
James	10000.0
Kim	45000.0
Mark	65000.0
Paul	20000.0
Teddy	20000.0

ID	NAME	AGE	ADDRESS	SALARY
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0
8	Paul	24	Houston	20000.0
9	James	44	Norway	5000.0
10	James	45	Texas	5000.0

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME ORDER BY NAME;
```

NAME	SUM(SALARY)
Allen	15000
David	85000
James	20000
Kim	45000
Mark	65000
Paul	40000
Teddy	20000

- **AND** & **OR** operators are used to compile multiple conditions to narrow down the selected data in an SQL statement. These two operators are called **conjunctive operators**.
- The AND operator allows the existence of multiple conditions in a SQLite statement's WHERE clause. [condition1] AND [condition2] will be true only when both condition1 and condition2 are true.

```
SELECT column1, column2, columnN FROM table_name WHERE [condition1] AND [condition2]...AND [conditionN];
```

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
3           Teddy       23          Norway      20000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
6           Kim         22          South-Hall  45000.0
7           James       24          Houston     10000.0
```

```
SELECT * FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;
```

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

- The OR operator is also used to combine multiple conditions in a SQL statement's WHERE clause. [condition1] OR [condition2] will be true if either condition1 or condition2 is true.

```
SELECT column1, column2, columnN FROM table_name WHERE [condition1] OR [condition2]...OR [conditionN]
```

```
SELECT * FROM COMPANY WHERE AGE >= 25 OR SALARY >= 65000;
```

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

Create the following tables

Consider the MOVIE DATABASE

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Actors

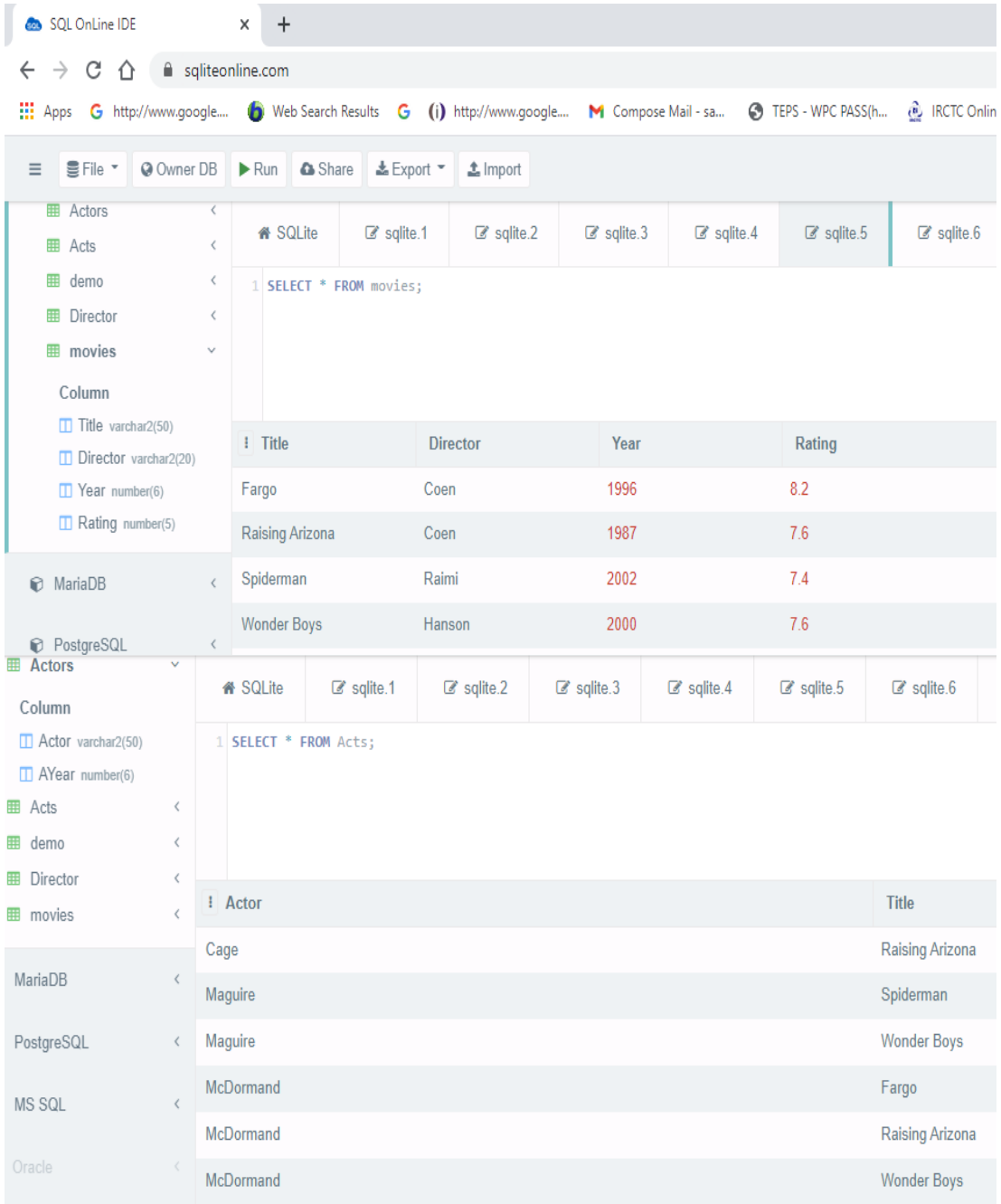
actor	ayear
Cage	1964
Hanks	1956
Maguire	1975
McDormand	1957

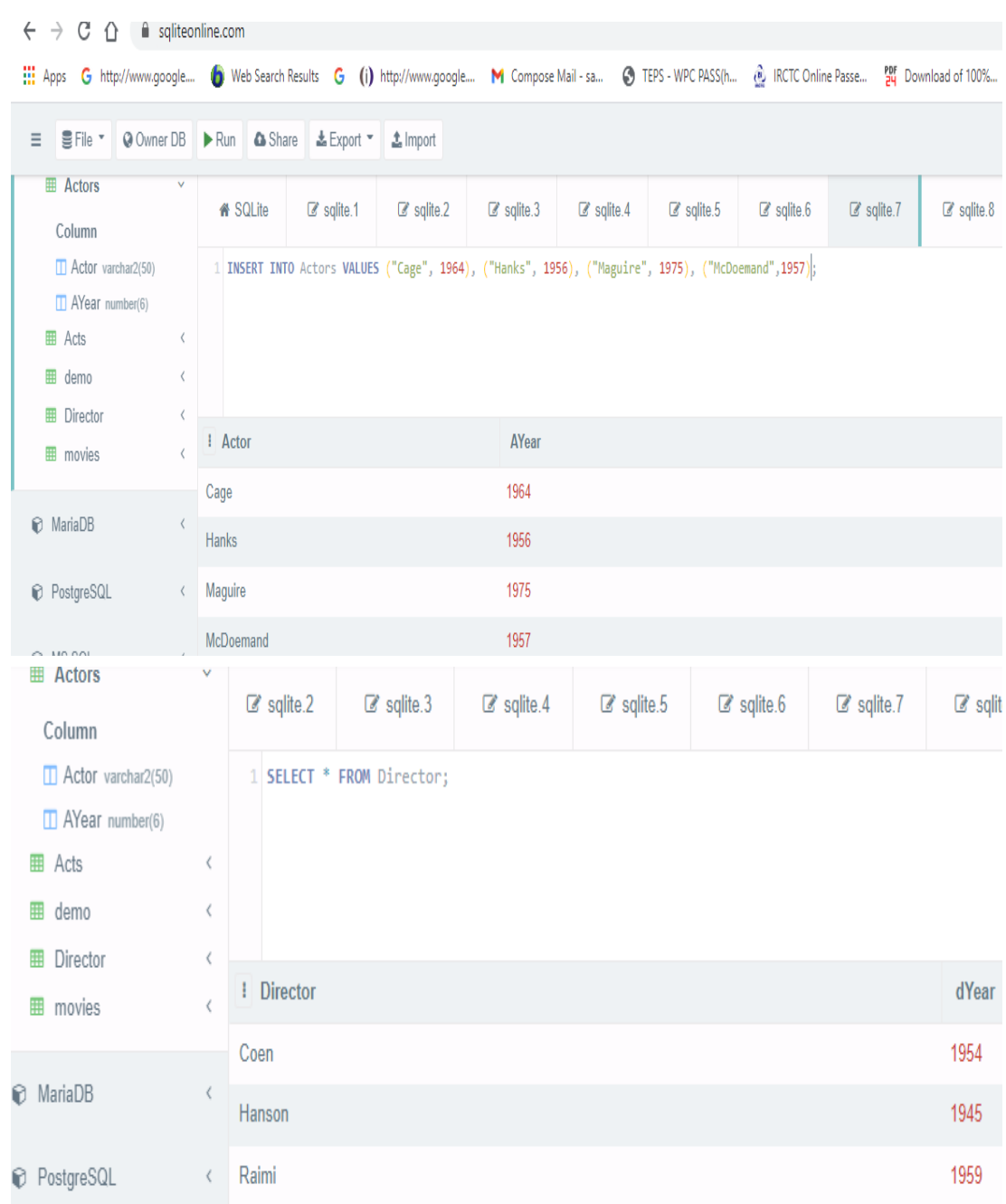
Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Directors

director	dyear
Coen	1954
Hanson	1945
Raimi	1959


SQL OnLine IDE
sqliteonline.com
Apps
Web Search Results
File
Owner DB
Run
Share
Export
Import
Actors
Acts
demo
Director
movies
Column
Title varchar2(50)
Director varchar2(20)
Year number(6)
Rating number(5)
MariaDB
PostgreSQL
MS SQL
Oracle
SQLite
sqlite.1
sqlite.2
sqlite.3
sqlite.4
sqlite.5
sqlite.6
SELECT * FROM movies;
Title
Director
Year
Rating
Fargo
Coen
1996
8.2
Raising Arizona
Coen
1987
7.6
Spiderman
Raimi
2002
7.4
Wonder Boys
Hanson
2000
7.6
SELECT * FROM Acts;
Actor
Title
Cage
Raising Arizona
Maguire
Spiderman
Maguire
Wonder Boys
McDormand
Fargo
McDormand
Raising Arizona
McDormand
Wonder Boys


sqliteonline.com
Apps
Web Search Results
File
Owner DB
Run
Share
Export
Import
Actors
Column
Actor varchar2(50)
AYear number(6)
Acts
demo
Director
movies
MariaDB
PostgreSQL
MS SQL
Oracle
SQLite
sqlite.1
sqlite.2
sqlite.3
sqlite.4
sqlite.5
sqlite.6
sqlite.7
sqlite.8
INSERT INTO Actors VALUES ("Cage", 1964), ("Hanks", 1956), ("Maguire", 1975), ("McDoemand",1957);
Actor
AYear
Cage
1964
Hanks
1956
Maguire
1975
McDoemand
1957
SELECT * FROM Director;
Director
dYear
Coen
1954
Hanson
1945
Raimi
1959

1. Find movies made after 1997

- select * from movies where year > 1997;

```
SELECT * FROM movies WHERE YEAR > 1997;
```

Title	Director	Year	Rating
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

2. Find movies made by Coen after 1980

- select * FROM movies where year > 1980 and Director = "Coen";

```
SELECT * FROM movies WHERE YEAR > 1980 AND Director = "Coen";
```

Title	Director	Year	Rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6

3. Find all movies and their ratings

- SELECT Title,Rating FROM movies;

```
1 SELECT Title,Rating  FROM movies;
```

Title	Rating
Fargo	8.2
Raising Arizona	7.6
Spiderman	7.4
Wonder Boys	7.6

4. Find all actors and directors

- SELECT Actor from Actors UNION SELECT Director from Director;

```
1 SELECT Actor FROM Actors UNION SELECT Director FROM Director;
```

Actor
Cage
Coen
Hanks
Hanson
Maguire
McDoemand
Raimi

5. Find Coen’s movies with McDormand

- SELECT Title from Acts WHERE Actor="McDormand" INTERSECT SELECT Title from movies WHERE Director = "Coen";

```
1 SELECT title FROM Acts WHERE Actor="McDormand" INTERSECT SELECT Title FROM movies WHERE Director = "Coen";
2
3
```

Title
Fargo
Raising Arizona

Distinct

```
SELECT * FROM student;
```

student_id	name	age
101	Adam	15
102	Ram	16
103	Alice	17
104	Wilson	15
105	Sam	16
106	Chris	Null
107	Alice	14
108	Sam	16
109	Nick	16
110	Peter	13

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT column1, column2, ... FROM table_name;

SELECT DISTINCT name from Student;

```
SELECT DISTINCT name FROM Student;
```

name
Adam
Ram
Alice
Wilson
Sam
Chris
Nick
Peter

Not Operator

- The NOT operator displays a record if the condition(s) is NOT TRUE.
- SELECT column1, column2, ... FROM table_name WHERE NOT condition;
- SELECT * FROM Student WHERE NOT name ='Alice' AND NOT name ='Sam';

```
1 SELECT * FROM Student
2 WHERE NOT name ='Alice' AND NOT name ='Sam';
```

student_id	name	age
101	Adam	15
102	Ram	16
104	Wilson	15
106	Chris	Null
109	Nick	16
110	Peter	13

NULL Value

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.
- A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- Have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names FROM table_name WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names FROM table_name WHERE column_name IS NOT NULL;
```

- SELECT * FROM Student WHERE age IS NULL;

```
1 SELECT * FROM Student WHERE age IS NULL;
```

student_id	name	age
106	Chris	Null

- SELECT * FROM Student WHERE age IS NOT NULL;

```
1 SELECT * FROM Student WHERE age IS NOT NULL;
```

student_id	name	age
101	Adam	15
102	Ram	16
103	Alice	17
104	Wilson	15
105	Sam	16
107	Alice	14
108	Sam	16
109	Nick	16
110	Peter	13

Update

- The UPDATE statement is used to modify the existing records in a table.
- UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
- UPDATE Student SET name = 'Sam David',age = '20' WHERE student_id = 105;

```
1 SELECT * FROM student;
```

student_id	name	age
101	Adam	15
102	Ram	16
103	Alice	17
104	Wilson	15
105	Sam	16
106	Chris	Null
107	Alice	14
108	Sam	16
109	Nick	16
110	Peter	13

```
1 UPDATE Student
2 SET name = 'Sam David',age = '20'
3 WHERE student_id = 105;
```

student_id	name	age
101	Adam	15
102	Ram	16
103	Alice	17
104	Wilson	15
105	Sam David	20
106	Chris	Null
107	Alice	14
108	Sam	16
109	Nick	16
110	Peter	13

TOP, LIMIT or ROWNUM Clause

- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

  Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

- SELECT * FROM Student LIMIT 3;

```
1 SELECT * FROM Student
2 LIMIT 3;
```

student_id	name	age
101	Adam	15
102	Ram	16
103	Alice	17

MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

MIN() Syntax

- SELECT MIN(column_name) FROM table_name WHERE condition;

MAX() Syntax

- SELECT MAX(column_name) FROM table_name WHERE condition;

SELECT student_id, name, MIN(age) FROM Student;

```
1 SELECT student_id, name, MIN(age) FROM Student;
```

student_id	name	MIN(age)
110	Peter	13

MAX() Functions

- SELECT student_id, name, Max(age) FROM Student;

```
1 SELECT student_id, name, Max(age) FROM Student;
```

student_id	name	Max(age)
105	Sam David	20

COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criterion.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.

COUNT() Syntax

- SELECT COUNT(column_name) FROM table_name WHERE condition;

AVG() Syntax

- SELECT AVG(column_name) FROM table_name WHERE condition;

SUM() Syntax

- SELECT SUM(column_name) FROM table_name WHERE condition;

SELECT COUNT(student_id) FROM Student;

```
1 SELECT COUNT(student_id)
2 FROM Student;
```

COUNT(student_id)
10

- SELECT AVG(age) FROM Student;

```
1 SELECT AVG(age)
2 FROM Student;
```

AVG(age)
15.777777777777779

- SELECT SUM(age) FROM Student;

```
1 SELECT SUM(age)
2 FROM Student;
```

SUM(age)
142

BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

- SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;

SELECT * FROM student WHERE age BETWEEN 16 AND 18;

```
1 SELECT * FROM student WHERE age BETWEEN 16 AND 18;
```

Student_Id	Name	Age
102	Ram	16
105	Alice	17
106	Sam	18
109	Chris	16

NOT BETWEEN Operator

- To display the products outside the range of the previous example, use NOT BETWEEN.
- SELECT * FROM student WHERE age NOT BETWEEN 16 AND 18;

```
1 SELECT * FROM student WHERE age NOT BETWEEN 16 AND 18;
```

Student_Id	Name	Age
101	Adam	15
103	Wilson	15
104	Adam	15
107	Peter	13
108	Sam David	20
110	Nick	19

LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- % - The percent sign represents zero, one, or multiple characters.
- _ - The underscore represents a single character.
- LIKE Syntax
- SELECT column1, column2, ... FROM table_name WHERE columnN LIKE pattern;

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

- SELECT * FROM student WHERE Name LIKE 'S%';

```
1 SELECT * FROM student WHERE Name LIKE 'S%';
```

Student_Id	Name	Age
106	Sam	18
108	Sam David	20

- SELECT * FROM student WHERE Name LIKE '%m';

```
1 SELECT * FROM student WHERE Name LIKE '%m';
```

Student_Id	Name	Age
101	Adam	15
102	Ram	16
104	Adam	15
106	Sam	18

Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of the query.

Alias Column Syntax

- SELECT column_name AS alias_name FROM table_name;

SELECT Student_Id AS ID, Name AS StudentName FROM student;

```
1 SELECT Student_Id AS ID, Name AS StudentName FROM student;
```

ID	StudentName
101	Adam
102	Ram
103	Wilson
104	Adam
105	Alice
106	Sam
107	Peter
108	Sam David
109	Chris
110	Nick

Alias for Tables

- selects all the orders from the customer with CustomerID=4 (Around the Horn).
- We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter).

```
SELECT o.OrderID, o.OrderDate, c.CustomerName FROM Customers AS c, Orders AS o WHERE
c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

Without using Alias

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName FROM Customers, Orders

WHERE Customers.CustomerName='Around the Horn' AND
Customers.CustomerID=Orders.CustomerID;
```

CASE Statement

- The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement).
- So, once a condition is true, it will stop reading and return the result.
- If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN conditionN THEN result
        ELSE result
END;
```

CASE Statement

```
SELECT orderid, quantity,
CASE
        WHEN quantity > 30 THEN 'The quantity is greater than 30’
        WHEN quantity = 30 THEN 'The quantity is 30’
        ELSE 'The quantity is under 30’
        END AS QuantityText
FROM orderdetails;
```

```
SELECT * FROM orderdetails;
```

snumber	orderid	productid	quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

```
SELECT orderid, quantity,
CASE
    WHEN quantity > 30 THEN 'The quantity is greater than 30'
    WHEN quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM orderdetails;
```

orderid	quantity	QuantityText
10248	12	The quantity is under 30
10248	10	The quantity is under 30
10248	5	The quantity is under 30
10249	9	The quantity is under 30
10249	40	The quantity is greater than 30

DELETE Statement

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

DELETE Syntax

- DELETE FROM table_name WHERE condition;

DELETE FROM orderdetails WHERE productid = 12;

```
2 SELECT * FROM orderdetails;
```

snumber	orderid	productid	quantity
1	10248	10	20
2	10248	2	5
3	10249	5	50
4	10250	9	75
5	10251	12	85

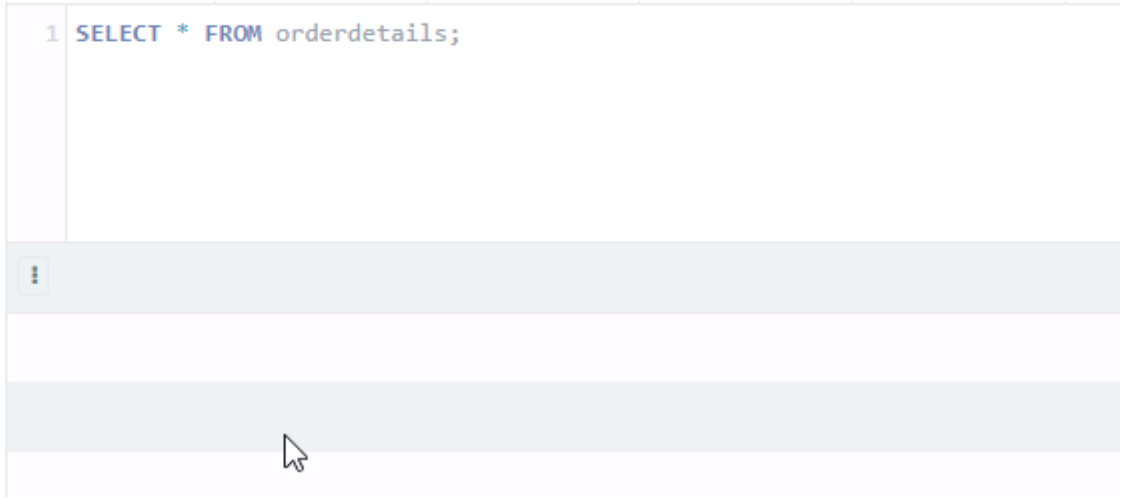
```
1 SELECT * FROM orderdetails;
```

snumber	orderid	productid	quantity
1	10248	10	20
2	10248	2	5
3	10249	5	50
4	10250	9	75

Delete All Records

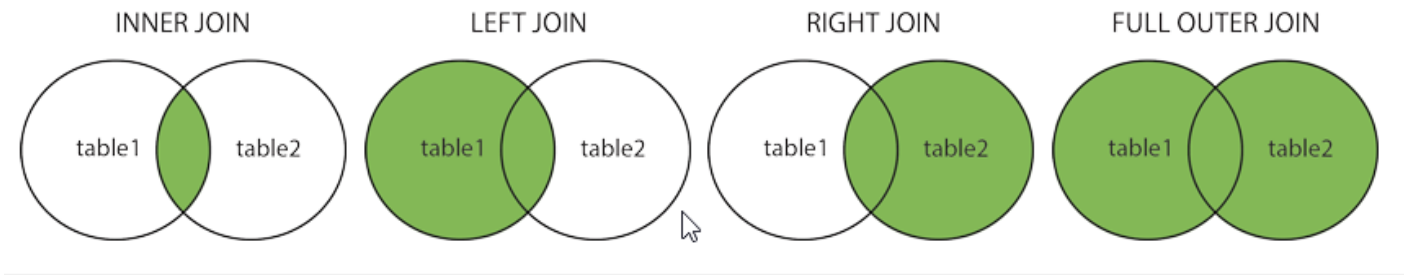
- It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact.
- DELETE FROM table_name;

DELETE FROM orderdetails;



Different Types of SQL JOINs

- **(INNER) JOIN**: Returns records that have matching values in both tables.
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table.
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table.
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table.



JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Orders Table

10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Customer Table

customerid	customername	contactname	country
1	Alfreds	Maria Anders	Germany
2	Nick John	Alice	Mexico
3	Antonio	Wilson Peter	Mexico

- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

```
1 SELECT orderdet.oderid, customers.customername, orderdet.orderdate
2 FROM orderdet
3 INNER JOIN customers ON orderdet.customerid=customers.customerid;
```

oderid	customername	orderdate
10308	Nick John	1996-09-18

LEFT JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

SELECT column_name(s) FROM table1 LEFT JOIN table2 ON table1.column_name = table2.column_name;

customer table

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com
6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

order table

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/05/01

SELECT customers.customer_id, orders.order_id, orders.order_date FROM customers LEFT JOIN orders ON customers.customer_id = orders.customer_id ORDER BY customers.customer_id;

Query Results

customer_id	order_id	order_date
4000	4	2016/04/20
5000	2	2016/04/18
6000	NULL	NULL
7000	1	2016/04/18
8000	3	2016/04/19
9000	NULL	NULL

RIGHT JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

```
SELECT column_name(s) FROM table1 RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

orders table

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/05/01

customers table

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com
6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

```
SELECT customers.customer_id, orders.order_id, orders.order_date  FROM customers  RIGHT JOIN orders  ON
customers.customer_id = orders.customer_id ORDER BY customers.customer_id;
```

Query Results

customer_id	order_id	order_date
NULL	5	2016/05/01
4000	4	2016/04/20
5000	2	2016/04/18
7000	1	2016/04/18
8000	3	2016/04/19

FULL OUTER JOIN

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;

Order table

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/05/01

Customer table

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com
6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

SELECT customers.customer_id, orders.order_id, orders.order_date FROM customers FULL OUTER JOIN orders ON customers.customer_id = orders.customer_id ORDER BY customers.customer_id;

Query Results

customer_id	order_id	order_date
NULL	5	2016/05/01
4000	4	2016/04/20
5000	2	2016/04/18
6000	NULL	NULL
7000	1	2016/04/18
8000	3	2016/04/19
9000	NULL	NULL

TCL -Transaction Control Languages

- **TCL** stands for **Transaction Control Languages**. These commands are used for maintaining consistency of the database and for the management of transactions made by the DML commands.

- A **Transaction** is a set of SQL statements that are executed on the data stored in DBMS.

- Whenever any transaction is made these transactions are temporarily happen in database. So, to make the changes permanent, we use **TCL** commands.

The TCL commands are

1. COMMIT
2. ROLLBACK
3. SAVEPOINT

COMMIT

- This command is used to save the data permanently.
- Whenever we perform any of the DDL command like -INSERT, DELETE or UPDATE, these can be rollback if the data is not stored permanently.
- So in order to be at the safer side COMMIT command is used.

Syntax

commit;

```
1  BEGIN TRANSACTION;
2  /* Create a table called NAMES */
3  CREATE TABLE NAMES(Id integer, Name varchar);
4  /* Create few records in this table */
5  INSERT INTO NAMES VALUES(1,'Tom');
6  INSERT INTO NAMES VALUES(2,'Lucy');
7  INSERT INTO NAMES VALUES(3,'Frank');
8  INSERT INTO NAMES VALUES(4,'Jane');
9  INSERT INTO NAMES VALUES(5,'Robert');
10 /* Display all the records from the table */
11 SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tom
2|Lucy
3|Frank
4|Jane
5|Robert
```

```
1  BEGIN TRANSACTION;
2  /* Create a table called NAMES */
3  CREATE TABLE NAMES(Id integer, Name varchar);
4  /* Create few records in this table */
5  INSERT INTO NAMES VALUES(1,'Tom');
6  INSERT INTO NAMES VALUES(2,'Lucy');
7  INSERT INTO NAMES VALUES(3,'Frank');
8  INSERT INTO NAMES VALUES(4,'Jane');
9  INSERT INTO NAMES VALUES(5,'Robert');
10 UPDATE NAMES
11 SET Name = "Sherlock"
12 WHERE Name = "Jane";
13 COMMIT;
14 /* Display all the records from the table */
15 SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tom
2|Lucy
3|Frank
4|Sherlock
5|Robert
```

ROLLBACK

- This command is used to get the data or restore the data to the last save point or last committed state.
- If due to some reasons the data inserted, deleted or updated is not correct, you can rollback the data to a particular save point or if save point is not done, then to the last committed state.

Syntax

rollback;

```
BEGIN TRANSACTION;
/* Create a table called NAMES */
CREATE TABLE NAMES(Id integer, Name varchar);
/* Create few records in this table */
INSERT INTO NAMES VALUES(1,'Tom');
INSERT INTO NAMES VALUES(2,'Lucy');
INSERT INTO NAMES VALUES(3,'Frank');
INSERT INTO NAMES VALUES(4,'Jane');
INSERT INTO NAMES VALUES(5,'Robert');
UPDATE NAMES
SET Name = "Sherlock"
WHERE Name = "Jane";
COMMIT;
UPDATE NAMES
SET Name = "Tim"
WHERE Name = "Tom";
/* Display all the records from the table */
SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tim
2|Lucy
3|Frank
4|Sherlock
5|Robert
```

SAVEPOINT

- This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

Syntax

Savepoint A;

```
BEGIN TRANSACTION;
/* Create a table called NAMES */
CREATE TABLE NAMES(Id integer, Name varchar);
/* Create few records in this table */
INSERT INTO NAMES VALUES(1,'Tom');
INSERT INTO NAMES VALUES(2,'Lucy');
INSERT INTO NAMES VALUES(3,'Frank');
INSERT INTO NAMES VALUES(4,'Jane');
INSERT INTO NAMES VALUES(5,'Robert');
INSERT into NAMES VALUES (6,'Jack');
Commit;
UPDATE NAMES
SET NAME= "Tim"
WHERE Id= 1;
SAVEPOINT A;

/* Display all the records from the table */
SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tim
2|Lucy
3|Frank
4|Jane
5|Robert
6|Jack
```

```
BEGIN TRANSACTION;
/* Create a table called NAMES */
CREATE TABLE NAMES(Id integer, Name varchar);
/* Create few records in this table */
INSERT INTO NAMES VALUES(1,'Tom');
INSERT INTO NAMES VALUES(2,'Lucy');
INSERT INTO NAMES VALUES(3,'Frank');
INSERT INTO NAMES VALUES(4,'Jane');
INSERT INTO NAMES VALUES(5,'Robert');
INSERT into NAMES VALUES (6,'Jack');
Commit;
UPDATE NAMES
SET NAME= "Tim"
WHERE Id= 1;
SAVEPOINT A;
INSERT INTO NAMES VALUES (7,"Zack");
Savepoint B;
INSERT INTO NAMES VALUES (8,"Bruno");
Savepoint C;
/* Display all the records from the table */
SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tim
2|Lucy
3|Frank
4|Jane
5|Robert
6|Jack
7|Zack
8|Bruno
```

```
BEGIN TRANSACTION;
/* Create a table called NAMES */
CREATE TABLE NAMES(Id integer, Name varchar);
/* Create few records in this table */
INSERT INTO NAMES VALUES(1,'Tom');
INSERT INTO NAMES VALUES(2,'Lucy');
INSERT INTO NAMES VALUES(3,'Frank');
INSERT INTO NAMES VALUES(4,'Jane');
INSERT INTO NAMES VALUES(5,'Robert');
INSERT into NAMES VALUES (6,'Jack');
Commit;
UPDATE NAMES
SET NAME= "Tim"
WHERE Id= 1;
SAVEPOINT A;
INSERT INTO NAMES VALUES (7,"Zack");
Savepoint B;
INSERT INTO NAMES VALUES (8,"Bruno");
Savepoint C;
Rollback to A;
/* Display all the records from the table */
SELECT * FROM NAMES;
```

```
$sqlite3 database.sdb < main.sql
1|Tim
2|Lucy
3|Frank
4|Jane
5|Robert
6|Jack
```

DCL - Data Control Language

- **DCL** stands for **Data Control Language** in Structured Query Language (SQL). As the name suggests these commands are used to control privilege in the database.

- The privileges (Right to access the data) are required for performing all the database operations like creating tables, views, or sequences.

- DCL command is a statement that is used to perform the work related to the rights, permissions, and other control of the database system.

There are two types of Privileges in database

- System Privilege
- Object Privilege

Commands in DCL

The two most important DCL commands are

- GRANT
- REVOKE

GRANT

- This command is used to grant permission to the user to perform a particular operation on a particular object.
- If you are a database administrator and you want to restrict user accessibility such as one who only views the data or may only update the data. You can give the privilege permission to the users.

Syntax

GRANT privilege_list ON Object_name TO user_name;

REVOKE

This command is used to take permission/access back from the user. If you want to return permission from the database that you have granted to the users at that time you need to run REVOKE command.

Syntax

REVOKE privilege_list ON object_name FROM user_name;

Following commands are granted to the user as a Privilege List

- EXECUTE
- UPDATE
- SELECT
- DELETE
- ALTER
- ALL

Advantages Of DCL commands

- It allows to restrict the user from accessing data in database.
- It ensures security in database when the data is exposed to multiple users.
- It is the wholesome responsibility of the data owner or data administrator to maintain the authority of grant and revoke privileges to the users preventing any threat to data.
- It prevents other users to make changes in database who have no access to Database