Controlled Redundancy: This is intentional redundancy used for optimizing database performance, such as storing pre-calculated totals to avoid recalculating them each time they are needed.
Uncontrolled Redundancy: This refers to unnecessary and unplanned duplication of data, often leading to inconsistencies and higher storage costs. For example, storing the same customer information in multiple tables without synchronization.
Two-Tier Architecture: In this model, the client handles both the presentation and application layers, and directly interacts with the database server (data layer). It's simpler but can be less efficient for complex applications.
Three-Tier Architecture: Adds an additional layer between the client and the database server, known as the application server or business logic layer. This structure improves scalability, maintainability, and the distribution of responsibilities.
Key: A minimal set of attributes necessary to uniquely identify a tuple in a relation.
Superkey: A set of one or more attributes that can uniquely identify a tuple, but is not necessarily minimal. For example, in a table with student ID and name, the student ID alone is a key, but student ID combined with name is a superkey.
Logical Structures: These are about how data is logically organized, independent of its physical storage. In databases, this includes schemas, tables, columns, indexes, views, and relationships. Logical structures represent the abstract design of the database, focusing on managing the data elements and their relationships.
Physical Structures: These refer to the physical storage of data on hardware. This includes file formats, data blocks, partitions, and the physical arrangement of data on the storage device. Physical structures are concerned with efficient storage, retrieval, and updating of data on physical media.
Why Duplicate Tuples are not Allowed in a Relation:In relational database theory, a relation is defined as a set of tuples. Sets, by definition, cannot have duplicate elements, which means a relation cannot have duplicate tuples. This ensures data integrity and uniqueness.
Definition and Use of Foreign Key: A foreign key is an attribute or set of attributes in one table that references the primary key in another table. It's used to establish and enforce a link between the data in two tables.
Difference between Transaction and Update: A transaction is a sequence of one or more operations (like updates, inserts, deletes) performed as a single logical unit of work. A transaction must be entirely completed or entirely failed, ensuring data integrity. An update, on the other hand, is just one type of operation that can be part of a transaction.
Reasons for NULL Values in Relations:
Attribute Not Applicable: The attribute does not apply to certain entities. For example, a 'spouse name' attribute for an unmarried person.
Unknown or Missing Information: The data for the attribute is not known or has not been provided.
Additional Functionality in N-Tier Architecture (n > 3): N-tier architectures can include additional layers like presentation, business, data access, and data layers, with potentially more layers for services like logging, error handling, and security. This enhances scalability, maintainability, and distribution of functionalities.

Ch 3 **class** is similar to an entity type in many ways. user-defined operations (or **transactions**) that will be applied to the database, including both retrievals and updates. to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. **logical design or data model** mapping; its result is a database
schema in the implementation data model of the DBMS. **Physical design** internal storage structures, file organizations, indexes, access paths, and physical design parameters for the files are specified. **Entity-** thing or object in the real world with an independent existence. **Attributes-** particular properties that describe it. **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. Attributes that are not divisible are called **simple or atomic attributes.** Most attributes have a single value for a particular entity; such attributes are called **single-valued**. A **multivalued** attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. The Age attribute is hence called a **derived** attribute and is said to be derivable from the Birth_date attribute, which is called a **stored** attribute. components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called **complex** attributes. An **entity type** defines a collection (or set) of entities that have the same attributes. The collection of all entities of a particular entity type in the database at any point in time is called an **entity set** or entity collection. An entity type describes the **schema or intension** for a set of entities that share the
same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the extension of the entity type. Each simple attribute of an entity type is associated with a **value set (or domain of values**), which specifies the set of values that may be assigned to that attribute for each individual entity. a **function** from E to the power set^6 P(V) of V:
$A : E \rightarrow P(V)$. Cartesian product of P(V1), P(V2), . . . , P(Vn), where V1, V2, . . . , Vn are the
value sets of the simple component attributes that form A: $V = P(P(V1) \times P(V2) \times \ldots \times P(Vn))$ The **power set P(V)** of a set V is the set of all subsets of V. **singleton** set is a set with only one element (value). A **relationship type** R among n entity types E1, E2, . . . , En defines a set of associations—or a **relationship set**—among entities from these entity types. the
relationship set R is a set of **relationship instances** ri, where each ri associates n individual entities (e1, e2, . . . , en), and each entity ej in ri is a member of entity set Ej, $1 \leq j \leq n$. Hence, a relationship set is a mathematical relation on E1, E2, . . . , En. Each of the entity types E1, E2, . . . , En is said to participate in the relationship type R; similarly, each of the individual entities e1, e2, . . . , en is said to participate in the relationship instance ri = (e1, e2, . . . , en). The **degree of a relationship** type is the number
of participating entity types. A relationship type of degree two is called **binary**, and one of degree three is
called **ternary**. This concept of representing relationship types as attributes is used in a class of data models called
functional data models. In relational databases, foreign keys are a type of reference attribute used to represent relationships. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means. the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships or self-referencing relationships**. The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. The **participation constraint** specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the **minimum cardinality constraint**. the participation of EMPLOYEE in WORKS_FOR is called **total participation**, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency**. Entity types that do not have key attributes of their own are called **weak entity types.** In contrast, regular entity types that do have a key attribute—which include all the examples discussed so far—are called strong entity types. We call this other entity type the **identifying or owner entity type**, and we call the relationship type that relates a **weak entity** type to its owner the **identifying relationship** of the weak entity type. A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. The identifying entity type is also sometimes called the **parent entity type or the dominant entity type**. The weak entity type is also sometimes called the **child entity type or the subordinate entity type.** The partial key is sometimes called the **discriminator**. A composite attribute is modeled as a **structured domain**, as illustrated by the Name attribute of EMPLOYEE. A multivalued attribute will generally be modeled as a separate class. Relationship types are called associations in UML terminology, and relationship instances are called links. A binary association (binary relationship type) is represented as a line connecting the participating classes (entity types), and may optionally have a name. A relationship attribute, called a link attribute, is placed in a box that is connected to the association's line by a dashed line. A recursive relationship type is called a **reflexive association** in UML, and the role names—like the multiplicities—are placed at the opposite ends of an association when compared with the placing of role names. Weak entities can be modeled using the UML construct called **qualified association(or qualified aggregation**).

Ch 5 A domain D is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A **data type or format** is also specified for each domain. A **relation schema** R, denoted by R(A1, A2, … , An), is made up of a relation name R and a list of attributes, A1, A2, … , An. Each **attribute** Ai is the name of a role
played by some domain D in the relation schema R. D is called the **domain of Ai** and is denoted by **dom(Ai)**. A relation schema is used to describe a relation; R is called the **name** of this relation. The **degree (or arity)** of a relation is the number of attributes n of its relation schema. A relation (or relation state) r(R) is a mathematical relation of degree n on the domains dom(A1), dom(A2), … , dom(An), which is a subset of the Cartesian product (denoted by ×) of the domains that define R: $r(R) \subseteq (dom(A1) \times dom(A2) \times \ldots \times (dom(An))$. total number of tuples in the Cartesian product is $|dom(A1)| \times |dom(A2)| \times \ldots \times |dom(An)|$.
Values and NULLs in the Tuples. Each value in a tuple is an **atomic** value; that is, it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed. This model is sometimes called the **flat relational model**. Much of the theory behind the relational model was developed with this assumption in mind, which is called the **first normal form** assumption. The relation schema can be interpreted as a declaration or a type of **assertion**. Each tuple in the relation can then be interpreted as a **fact or a particular instance** of the assertion. **predicate**; in this case,

the values in each tuple are interpreted as values that satisfy the predicate. Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL. We call these **schema-based constraints or explicit constraints.** Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way. We call these application-based or **semantic constraints** or business rules. **Domain constraints** specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double-precision float). Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and other special data types. Domains can also be described by a subrange of values from a data type or as an enumerated data type in which all possible values are explicitly listed. for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that: $t1[SK] \neq t2[SK]$. Any such set of attributes SK is called a superkey of the relation schema R. A **super-key** SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK. It is common to designate one of the candidate keys as the **primary key** of the relation. This is the candidate key whose values are used to identify tuples in the relation. A **relational database schema** S is a set of relation schemas S = {R1, R2, … , Rm} and a set of **integrity constraints** IC. A **relational database state** DB of S is a set of relation states DB = {r1, r2, … , rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC. A **relational database state** is sometimes called a relational database snapshot or instance. The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2. State constraints are sometimes called **static constraints**, and transition constraints are sometimes called **dynamic constraints**. A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

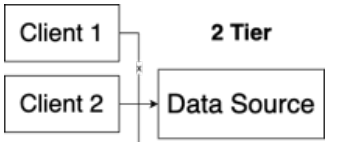
Ch 6. An SQL schema is identified by a schema name and includes an **authorization identifier** to indicate the user or account who owns the schema, as well as **descriptors** for each element in the schema. Schema elements include tables, types, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. they can generally be used in string comparisons by being cast (or **coerced or converted**) into the equivalent strings.
multiset;
duplicate tuples can appear more than once in a table, and in the result of a query.
SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND Lname = 'Smith')
UNION (SELECT DISTINCT Pnumber FROM PROJECT, WORKS_ON, EMPLOYEE WHERE Pnumber = Pno AND Essn = Ssn AND Lname = 'Smith');

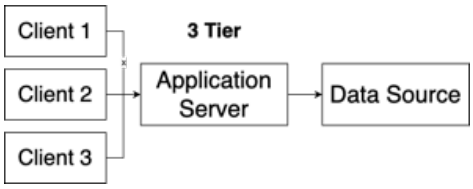
BASIC SQL COMMANDS

1. CREATE: CREATE TABLE table_name (column1 datatype, column2 datatype, ...);
2. INSERT: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
3. UPDATE: UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
4. DELETE: DELETE FROM table_name WHERE condition;
5. ALTER: ALTER TABLE table_name ADD column_name datatype; ALTER TABLE table_name DROP COLUMN column_name;

ORDER, GROUP, LIKE, IN, WHERE

1. ORDER BY: SELECT * FROM table_name ORDER BY column1 ASCIDESC;
2. GROUP BY: SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING condition;
3. LIKE (starts with): SELECT * FROM table_name WHERE column_name LIKE 'prefix%';
4. LIKE (ends with): SELECT * FROM table_name WHERE column_name LIKE '%suffix';
5. LIKE (contains): SELECT * FROM table_name WHERE column_name LIKE '%substring%';
6. NOT LIKE (starts with): SELECT * FROM table_name WHERE column_name NOT LIKE 'prefix%';
7. NOT LIKE (ends with): SELECT * FROM table_name WHERE column_name NOT LIKE '%suffix';
8. NOT LIKE (contains): SELECT * FROM table_name WHERE column_name NOT LIKE '%substring%';
9. IN: SELECT * FROM table_name WHERE column_name IN (value1, value2, ...);
10. NOT IN: SELECT * FROM table_name WHERE column_name NOT IN (value1, value2, ...);
11. WHERE: SELECT * FROM table_name WHERE condition;
12. BETWEEN: SELECT * FROM table_name WHERE column_name BETWEEN value1 AND value2;
13. IS NULL: SELECT * FROM table_name WHERE column_name IS NULL;
14. IS NOT NULL: SELECT * FROM table_name WHERE column_name IS NOT NULL;
15. AND: SELECT * FROM table_name WHERE condition1 AND condition2;
16. OR: SELECT * FROM table_name WHERE condition1 OR condition2;
17. LIMIT: SELECT * FROM table_name LIMIT 10;
18. DISTINCT: SELECT DISTINCT(column_name) FROM table_name;
19. JOIN: SELECT * FROM table1 INNER JOIN table2 ON table1.column = table2.column;
20. LEFT JOIN: SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
21. RIGHT JOIN: SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
22. FULL OUTER JOIN: SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.column = table2.column
23. SELECT Ename, (SELECT AVG(Salary) FROM EMPLOYEE) AS Average_Salary FROM EMPLOYEE WHERE Dno = 5;

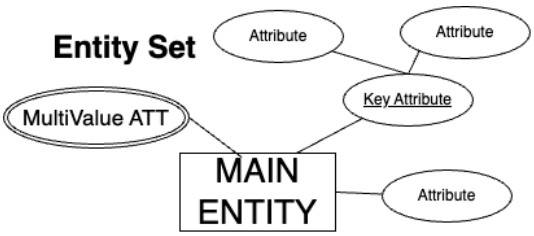




MATH FUNCTIONS

1. MAX: SELECT MAX(column_name) FROM table_name;
2. MIN: SELECT MIN(column_name) FROM table_name;
3. AVG: SELECT AVG(column_name) FROM table_name;

COUNT: SELECT COUNT(*) FROM table_name;
MULTIPLICATION: SELECT column1 * column2 AS result FROM table_name;
ADDITION: SELECT column1 + column2 AS result FROM table_name;
SUBTRACTION: SELECT column1 - column2 AS result FROM table_name;



CARTESIAN PRODUCT

CROSS JOIN: SELECT * FROM table1 CROSS JOIN table2;
Formula: R(A1, A2, ..., An) = dom(A1) x dom(A2) x ... x dom(An)

Cardinality 1:1 – One to One / 1:N – One to Many / N:1 – Many to One / N:M (N:N) – Many to Many
ID (PK), Name, Key (Attributes) -> Employee (Entity) ^{1} -> ^{1} Manage => Factory <- City
Employee (Entity) ^{N} -> ^{1} Works At -> Factory