A ***database application*** is software that helps business users interact with database systems.

Database roles
People interact with databases in a variety of roles:

- A ***database administrator*** is responsible for securing the database system against unauthorized users. A database administrator enforces procedures for user access and database system availability.
- **Authorization**. Many database users should have limited access to specific tables, columns, or rows of a database. Database systems authorize individual users to access specific data.
- **Rules**. Database systems ensure data is consistent with structural and business rules. Ex: When multiple copies of data are stored in different locations, copies must be synchronized as data is updated. Ex: When a course number appears in a student registration record, the course must exist in the course catalog.
- The ***query processor*** interprets queries, creates a plan to modify the database or retrieve data, and returns query results to the application. The query processor performs ***query optimization*** to ensure the most efficient instructions are executed on the data.
- The ***storage manager*** translates the query processor instructions into low-level file-system commands that modify or retrieve data. Database sizes range from megabytes to many terabytes, so the storage manager uses ***indexes*** to quickly locate data.
- The ***transaction manager*** ensures transactions are properly executed. The transaction manager prevents conflicts between concurrent transactions. The transaction manager also restores the database to a consistent state in the event of a transaction or system failure.

MongoDB	MongoDB	NoSQL	Open source	5

- ***INSERT*** inserts rows into a table.
- ***SELECT*** retrieves data from a table.
- ***UPDATE*** modifies data in a table.
- ***DELETE*** deletes rows from a table.

The SQL ***CREATE TABLE*** statement creates a new table by specifying the table and column names. Each column is assigned a ***data type*** that indicates the format of column values. Data types can be numeric, textual, or complex. Ex:

- INT stores integer values.
- DECIMAL stores fractional numeric values.
- VARCHAR stores textual values.
- DATE stores year, month, and day.

- Analysis
- Logical design
- Physical design

The ***analysis*** phase specifies database requirements without regard to a specific database system. Requirements are represented as entities, relationships, and attributes. An entity is a person, place, activity, or thing. A relationship is a link between entities, and an attribute is a descriptive property of an entity.

Analysis *has many alternative names, such as conceptual design, entity-relationship modeling, and requirements definition.*

Logical design
The ***logical design*** phase implements database requirements in a specific database system. For relational database systems, logical design converts entities, relationships, and attributes into tables, keys, and columns.

Physical design
The ***physical design*** phase adds indexes and specifies how tables are organized on storage media. Physical design affects query processing speed but never affects the query result. The principle that physical design never affects query results is called ***data independence***.

To simplify the use of SQL with a general-purpose language, database programs typically use an application programming interface. An ***application programming interface***, or ***API***

MySQL Command-Line Client
The ***MySQL Command-Line Client*** is a text interface included in the MySQL Server download. MySQL Server returns an ***error code*** and description when an SQL statement is syntactically incorrect or the database cannot execute the statement.

. A ***tuple*** is an ordered collection of elements enclosed in parentheses. Ex: **(a, b, c)** and **(c, b, a)** are different, since tuples are ordered.

The data structure organizes data in tables:

- A ***table*** has a name, a fixed tuple of columns, and a varying set of rows.
- A ***column*** has a name and a data type.
- A ***row*** is an unnamed tuple of values. Each value corresponds to a column and belongs to the column's data type.
- A ***data type*** is a named set of values, from which column values are drawn.

Since a table is a set of rows, the rows have no inherent order.

- Synonyms:
- Table, File, Relation
- Row, Record, Tuple
- Column, Field, Attribute

Business rules are based on business policy and specific to a particular database

Type	Description	Examples

Literals	Explicit values that are string, numeric, or binary.
	Strings must be surrounded by single quotes or double quotes.
	Binary values are represented with `x'0'` where the 0 is any hex value.
		`'String'`
		`"String"`
		`123`
		`x'0fa2'`

Keywords	Words with special meaning.	SELECT, FROM, WHERE
Identifiers	Objects from the database like tables, columns, etc.	City, Name, Population

Comments	Statement intended only for humans and ignored by the database when parsing an SQL statement.	`-- single line comment`
		`/* multi-line`
		`   Comment */`

SQL sublanguages
The SQL language is divided into five sublanguages:

- ***Data Definition Language*** (DDL) defines the structure of the database.
- ***Data Query Language*** (DQL) retrieves data from the database.
- ***Data Manipulation Language*** (DML) manipulates data stored in a database.
- ***Data Control Language*** (DCL) controls database user access.
- ***Data Transaction Language*** (DTL) manages database transactions.

All data in a relational database is structured in tables:

- A ***table*** has a name, a fixed sequence of columns, and a varying set of rows.
- A ***column*** has a name and a data type.
- A ***row*** is an unnamed sequence of values. Each value corresponds to a column and belongs to the column's data type.
- A ***cell*** is a single column of a single row.

A table must have at least one column but any number of rows. A table without rows is called an ***empty table***.

Rule 7 is called ***data independence***. Data independence allows database administrators to improve query performance by changing the organization of data on storage devices, without affecting query results.

CREATE TABLE and DROP TABLE statements
The ***CREATE TABLE*** statement creates a new table by specifying the table name, column names, and column data types. Example data types are:

- **INT or INTEGER** — integer values
- **VARCHAR(N)** — values with 0 to N characters
- **DATE** — date values
- **DECIMAL(M, D)** — numeric values with M digits, of which D digits follow the decimal point

The ***DROP TABLE*** statement deletes a table, along with all the table's rows, from a database.

ALTER TABLE statement
The ***ALTER TABLE*** statement adds, deletes, or modifies columns on an existing table.

A ***data type*** is a named set of values from which column values are drawn. In relational databases, most data types fall into one of the following categories:

- ***Integer*** data types represent positive and negative integers. Several integer data types exist, varying by the number of bytes allocated for each value. Common integer data types include INT, implemented as 4 bytes of storage, and SMALLINT, implemented as 2 bytes.

Category	Example	Data type	Storage	Notes
Integer	34 and -739448	TINYINT	1 byte	Signed range: -128 to 127 Unsigned range: 0 to 255
		SMALLINT	2 bytes	Signed range: -32,768 to 32,767 Unsigned range: 0 to 65,535
		MEDIUMINT	3 bytes	Signed range: -8,388,608 to 8,388,607 Unsigned range: 0 to 16,777,215
		INTEGER or INT	4 bytes	Signed range: -2,147,483,648 to 2,147,483,647 Unsigned range: 0 to 4,294,967,295
		BIGINT	8 bytes	Signed range: -2^{63} to 2^{63} -1 Unsigned range: 0 to 2^{64} -1

Type	Operator	Description	Example	Value
Arithmetic	+	Adds two numeric values	4 + 3	7
	- (unary)	Reverses the sign of one numeric value	-(-2)	2
	- (binary)	Subtracts one numeric value from another	11 - 5	6

	*	Multiplies two numeric values	`3 * 5`	`15`
	/	Divides one numeric value by another	`4 / 2`	`2`
	% (modulo)	Divides one numeric value by another and returns the integer remainder	`5 % 2`	`1`
	^	Raises one numeric value to the power of another	`5^2`	`25`
Comparison	=	Compares two values for equality	`1 = 2`	`FALSE`
	!=	Compares two values for inequality	`1 != 2`	`TRUE`
	<	Compares two values with <	`2 < 2`	`FALSE`
	<=	Compares two values with ≤	`2 <= 2`	`TRUE`
	>	Compares two values with >	`'2019-08-13' > '2021-08-13'`	`FALSE`

UPDATE statement
The ***UPDATE*** statement modifies existing rows in a table. The UPDATE statement uses the ***SET*** clause to specify the new column values. An optional WHERE clause specifies which rows are updated. Omitting the WHERE clause results in all rows being updated.

DELETE statement
The ***DELETE*** statement deletes existing rows in a table. The ***FROM*** keyword is followed by the table name whose rows are to be deleted. An optional WHERE clause specifies which rows should be deleted. Omitting the WHERE clause results in all rows in the table being deleted.

*The **TRUNCATE** statement deletes all rows from a table. TRUNCATE is nearly identical to a DELETE statement with no WHERE clause except for minor differences that depend on the database system.*

The **MERGE** statement selects data from one table, called the source, and inserts the data to another table, called the target.

Primary keys
A ***primary key*** is a column, or group of columns, used to identify a row. The primary key is usually the table's first column and appears on the left of table diagrams, but the position is not significant to the database.

A ***simple primary key*** consists of a single column. A ***composite primary key*** consists of multiple columns. Auto-increment columns
An ***auto-increment column*** is a numeric column that is assigned an automatically incrementing value when a new row is inserted.

Database users occasionally make the following errors when inserting primary keys:

- Inserting values for auto-increment primary keys.
- Omitting values for primary keys that are not auto-increment columns.

MySQL allows insertion of a specific value to an auto-increment column. However, overriding auto-increment for a primary key is usually a mistake.

A ***foreign key*** is a column, or group of columns, that refer to a primary key. The data types of the foreign and primary keys must be the same, but the names may be different.

FOREIGN KEY constraint
A foreign key constraint is added to a CREATE TABLE statement with the ***FOREIGN KEY*** and ***REFERENCES*** keywords. When a foreign key constraint is specified, the database rejects insert, update, and delete statements that violate referential integrity.

Referential integrity actions
An insert, update, or delete that violates referential integrity can be corrected manually. However, manual corrections are time-consuming and error-prone. Instead, databases automatically correct referential integrity violations with any of four actions, specified as SQL constraints:

- ***RESTRICT*** rejects an insert, update, or delete that violates referential integrity.
- ***SET NULL*** sets invalid foreign keys to NULL.
- ***SET DEFAULT*** sets invalid foreign keys to the foreign key default value.
- ***CASCADE*** propagates primary key changes to foreign keys.

Column and table constraints
A ***constraint*** is a rule that governs allowable values in a database. Constraints are based on relational and business rules, and implemented with special keywords in a CREATE TABLE statement. The database automatically rejects insert, update, and delete statements that violate a constraint.

Adding and dropping constraints
Constraints are added and dropped with the `ALTER TABLE TableName` followed by an ADD, DROP, or CHANGE clause

values.

BETWEEN operator

The ***BETWEEN*** operator provides an alternative way to determine if a value is between two other values. The operator is written `value BETWEEN minValue AND maxValue` and is equivalent to `value >= minValue AND value <= maxValue`.

LIKE operator

The ***LIKE*** operator, when used in a WHERE clause, matches text against a pattern using the two wildcard characters % and _.

- % matches any number of characters. Ex: `LIKE 'L%t'` matches "Lt", "Lot", "Lift", and "Lol cat".
- _ matches exactly one character. Ex: `LIKE 'L_t'` matches "Lot" and "Lit" but not "Lt" and "Loot".
- The ***ORDER BY*** clause orders selected rows by one or more columns in ascending (alphabetic or increasing) order. The ***DESC*** keyword with the ORDER BY clause orders rows in descending order.

Function	Description	Example
ABS(n)	Returns the absolute value of *n*	`SELECT ABS(-5);` returns 5

LOWER(s)	Returns the lowercase s	`SELECT LOWER('MySQL');`
		`returns 'mysql'`

TRIM(s)	Returns the string s without leading and trailing spaces	`SELECT TRIM('   test   ');`
		`returns 'test'`

HOUR(t)
MINUTE(t)
SECOND(t)
	Returns the hour, minute, or second from time t	`SELECT HOUR('22:11:45');`
		`returns 22`
		`SELECT MINUTE('22:11:45');`
		`returns 11`
		`SELECT SECOND('22:11:45');`
		`returns 45`

Aggregate functions
An ***aggregate function*** processes values from a set of rows and returns a summary value. Common aggregate functions are:

- ***COUNT()*** counts the number of rows in the set.
- ***MIN()*** finds the minimum value in the set.
- ***MAX()*** finds the maximum value in the set.
- ***SUM()*** sums all the values in the set.
- ***AVG()*** computes the arithmetic mean of all the values in the set.

Aggregate functions appear in a SELECT clause and process all rows that satisfy the WHERE clause condition. If a SELECT statement has no WHERE clause, the aggregate function processes all rows.

HAVING clause
The *HAVING* clause is used with the GROUP BY clause to filter group results. The optional HAVING clause follows the GROUP BY clause and precedes the optional ORDER BY clause.

A ***join*** is a SELECT statement that combines data from two tables, known as the ***left table*** and ***right table***, into a single result. The tables are combined by comparing columns from the left and right tables, usually with the = operator. The columns must have comparable data types.

a column name can be replaced with an alias. The alias follows the column name, separated by an optional ***AS*** keyword.

A ***join clause*** determines how a join query handles unmatched rows. Two common join clauses are:

- ***INNER JOIN*** selects only matching left and right table rows.
- ***FULL JOIN*** selects all left and right table rows, regardless of match.

In a FULL JOIN result table, unmatched left table rows appear with NULL values in right table columns, and vice versa.

- ***LEFT JOIN*** selects all left table rows, but only matching right table rows.
- ***RIGHT JOIN*** selects all right table rows, but only matching left table rows.

An ***outer join*** is any join that selects unmatched rows, including left, right, and full joins.

The ***UNION*** keyword combines the two results into one table.

Equijoins
An ***equijoin*** compares columns of two tables with the = operator. Most joins are equijoins. A ***non-equijoin*** compares columns with an operator other than =, such as < and >.

Self-joins
A ***self-join*** joins a table to itself.

Cross-joins
A ***cross-join*** combines two tables without comparing columns. A cross-join uses a ***CROSS JOIN*** clause without an ON clause. As a result, all possible combinations of rows from both tables appear in the result.

A ***subquery***, sometimes called a ***nested query*** or ***inner query***, is a query within another SQL query.

. An ***alias*** is a temporary name assigned to a column or table. The ***AS*** keyword follows a column or table name to create an alias.

In some databases, view data can be stored. A ***materialized view*** is a view for which data is stored at all times. Whenever a base table changes, the corresponding view tables can also change, so materialized views must be refreshed.

When ***WITH CHECK OPTION*** is specified, the database rejects inserts and updates that do not satisfy the view query WHERE clause. Instead, the database generates an error message that explains the violation.

An ***entity-relationship model*** is a high-level representation of data requirements, ignoring implementation details

An entity-relationship model includes three kinds of objects:

- An ***entity*** is a person, place, product, concept, or activity.
- A ***relationship*** is a statement about two entities.
- An ***attribute*** is a descriptive property of an entity.

A relationship is usually a statement about two different entities, but the two entities may be the same. A ***reflexive relationship*** relates an entity to itself.

An ***entity-relationship diagram***, commonly called an ***ER diagram***, is a schematic picture of entities, relationships, and attributes. Entities are drawn as rectangles.

Types and instances
In entity-relationship modeling, a type is a set:

- An ***entity type*** is a set of things. Ex: All employees in a company.
- A ***relationship type*** is a set of related things. Ex: Employee-Manages-Department is a set of (employee, department) pairs, where the employee manages the department.
- An ***attribute type*** is a set of values. Ex: All employee salaries.

Entity, relationship, and attribute types usually become tables, foreign keys, and columns, respectively.
An instance is an element of a set:

- An ***entity instance*** is an individual thing. Ex: The employee Sam Snead.
- A ***relationship instance*** is a statement about entity instances. Ex: "Maria Rodriguez manages Sales."
- An ***attribute instance*** is an individual value. Ex: The salary $35,000.
- ***Analysis*** develops an entity-relationship model, capturing data requirements while ignoring implementation details.

- ***Logical design*** converts the entity-relationship model into tables, columns, and keys for a particular database system.
- ***Physical design*** adds indexes and specifies how tables are organized on storage media.

Analysis steps

Step	Name
1	Discover entities, relationships, and attributes
2	Determine cardinality
3	Distinguish strong and weak entities
4	Create supertype and subtype entities

Logical design steps

Step	Name
5	Implement entities
6	Implement relationships
7	Implement attributes
8	Apply normal form

cardinality refers to maxima and minima of relationships and attributes.

Relationship maximum is the greatest number of instances of one entity that can relate to a single instance of another entity.

Relationship minimum is the least number of instances of one entity that can relate to a single instance of another entity.

In ER diagrams, attribute maximum and minimum follow the attribute name. The minimum appears in parentheses.

A ***subtype entity*** is a subset of another entity type, called the ***supertype entity***. Ex: Managers are a subset of employees, so Manager is a subtype entity of the Employee supertype entity. On ER diagrams, subtype entities are drawn within the supertype. Vehicle Supertype entity and “car” “truck” “motorcycle” subtype

The identifying relationship is called an ***IsA relationship***.

Partitions
A ***partition*** of a supertype entity is a group of mutually exclusive subtype entities.

After entities, relationships, attributes, cardinality, and strong and weak entities are determined, the database designer looks for supertype and subtype entities.

Creating supertype and subtype entities is the last of four analysis steps:

1. Discover entities, relationships, and attributes
2. Determine cardinality
3. Distinguish strong and weak entities
4. Create supertype and subtype entities

Logical design follows analysis. Logical design converts an entity-relationship model to tables, columns, and keys for a specific database system.

. One popular convention, called ***crow's foot notation***, depicts cardinality as a circle (zero), a short line (one), or three short lines (many). The three short lines look like a bird's foot, hence the name "crow's foot notation".

An **intangible entity** is documented in the data model, but not tracked with data in the database.

PRIMARY KEYS:

- **Stable**. Primary key values should not change. When a primary key value changes, statements that specify the old value must also change. Furthermore, the new primary key value must cascade to matching foreign keys.
- **Simple**. Primary key values should be easy to type and store. Small values are easy to specify in an SQL WHERE clause and speed up query processing. Ex: A 2-byte integer is easier to type and faster to process than a 15-byte character string.
- **Meaningless**. Primary keys should not contain descriptive information. Descriptive information occasionally changes, so primary keys containing descriptive information are unstable.

An ***artificial key*** is a single-column primary key created by the database designer when no suitable single-column or composite primary key exists. Usually artificial key values are integers, generated automatically by the database as new rows are inserted to the table. Artificial keys are stable, simple, and meaningless.

Dependence of one column on another is called ***functional dependence***

Redundancy is the repetition of related values in a table.

Normal forms are rules for designing tables with less redundancy

A ***candidate key*** is a simple or composite column that is unique and minimal. ***Minimal*** means all columns are necessary for uniqueness. A table may have several candidate keys. The database designer designates one candidate key as the primary key.

A ***non-key*** column is a column that is not contained in a candidate key.

A table is in ***third normal form*** if, whenever a non-key column A depends on column B, then B is unique. Columns A and B may be simple or composite. Although B is unique, B is not necessarily minimal and therefore is not necessarily a candidate key.

A table is in ***Boyce-Codd normal form*** if, whenever column A depends on column B, then B is unique. Columns A and B may be simple or composite. This definition is identical to the definition of third normal form with the term 'non-key' removed. “GOLD STANDARD”

Boyce-Codd normal form is ideal for tables with frequent inserts, updates, and deletes.

Trivial dependencies

*When the columns of A are a subset of the columns of B, A always depends on B. Ex: FareClass depends on (FlightCode, FareClass). These dependencies are called **trivial**.*

This redundancy is eliminated with normalization, the last step of logical design.

Normalization eliminates redundancy by decomposing a table into two or more tables in higher normal form.

Denormalization means intentionally introducing redundancy by merging tables

Databases commonly support four alternative table structures:

- Heap table In a ***heap table***, no order is imposed on rows.
    - Heap tables optimize insert operations. Heap tables are particularly fast for bulk load of many rows, since rows are stored in load order
- Sorted table In a ***sorted table***, the database designer identifies a ***sort column*** that determines physical row order.
- Hash table In a ***hash table***, rows are assigned to buckets. A ***bucket*** is a block or group of blocks containing rows. The ***modulo function*** is a simple hash function with four steps:
- Table cluster ***Table clusters***, also called ***multi-tables***, interleave rows of two or more tables in the same storage area.

- A ***table scan*** is a database operation that reads table blocks directly, without accessing an index.
- An ***index scan*** is a database operation that reads index blocks sequentially, in order to locate the needed table blocks.

Hit ratio, also called ***filter factor*** or ***selectivity***, is the percentage of table rows selected by a query. When a SELECT query is executed
In a ***binary search***, the database repeatedly splits the index in two until it finds the entry containing the search value:

Indexes may also be dense or sparse:

- A ***dense index*** contains an entry for every table row.
- A ***sparse index*** contains an entry for every table block.

Hash index

- Bitmap index
- Logical index
- Function index

In a ***hash index***, index entries are assigned to buckets. A ***bitmap index*** is a grid of bits:

Bitmap indexes contain ones and zeros.

A ***tablespace*** is a database object that maps one or more tables to a single file CREATE TABLESPACE

Logical design specifies tables, columns, and keys. ***Physical design*** specifies indexes, table structures, and partitions. Physical design affects query performance but never affects query results. **A *storage engine* or *storage manager* translates instructions generated by a query processor into low-level commands that access data on storage media.**

Statement	Description	Syntax
CREATE INDEX	Create an index	`CREATE INDEX IndexName ON TableName (Column1, Column2, ..., ColumnN);`