**- Data integrity :** all data is correct, up to date and not repeated. there are 3 types :

Entity : every row be unique .

Referential : related tables must be connected

Domain : acceptable values for columns

- **Schema** : the way that database structured . there are 3 types :

Conceptual / High level : how data is related .

Logical : structure of table

Physical : implementation of database, how to access it and type of tables.

- **Relationships** : 1 : 1 /1 : M / M : M

- **Design of relationships**:

**1 : 1** => attribute

**1 : M** => foreign key from parent to children

**M : M** => make third table called junction

- **Keys :** unique, never change and never NULL .

**- Why Keys ? to :**

• Protect integrity • Make every row unique

• Improve functionality • Less work

• Allows for added complexity

- **Super key :** Any number of columns forces every row to talk about one entity.

( how every row be unique ? )

- **Candidate key :** The least number of columns make row unique . ( how many columns are needed ? )

**SQL** is comprised of different classes of commands:

1. Data Manipulation Language (DML): SELECT, INSERT, UPDATE, and DELETE statements.
2. Data Definition Language (DDL): Schema definitions for tables, indexes, views, and other objects.
3. Data Control Language (DCL): Security, access controls.

* **Joins** : Combines columns from one or more tables and produces a new table. Used to express queries that involve data that spans multiple tables.
* **Aggregates functions :**

• AVG(COL): The average of the values in COL

• MIN(COL): The minimum value in COL

• MAX(COL): The maximum value in COL

• COUNT(COL): The number of tuples in the relation

* **The following three queries are equivalent:**

• SELECT COUNT(\*) FROM student WHERE login LIKE '%@cs';

• SELECT COUNT(login) FROM student WHERE login LIKE '%@cs';

• SELECT COUNT(1) FROM student WHERE login LIKE '%@cs';

**- Some aggregate functions support the DISTINCT keyword:**

• SELECT COUNT(DISTINCT login) FROM student WHERE login LIKE '%@cs';

- Output of other columns outside of an aggregate is undefined (e.cid is undefined below):

• SELECT AVG(s.gpa), e.cid FROM enrolled AS e, student AS s WHERE e.sid = s.sid;

- Non-aggregated values in SELECT output clause must appear in GROUP BY clause:

• SELECT AVG(s.gpa), e.cid FROM enrolled AS e, student AS s

WHERE e.sid = s.sid

GROUP BY e.cid;

- The HAVING clause filters output results based on aggregation computation. This make HAVING behave like a WHERE clause for a GROUP BY :

• SELECT AVG(s.gpa) AS avg\_gpa, e.cid FROM enrolled AS e, student AS s

WHERE e.sid = s.sid

GROUP BY e.cid

HAVING avg\_gpa > 3.9;

- The above query syntax is supported by many major database systems, but is not compliant with the SQL standard. To make the query standard compliant, we must repeat use of AVG(S.GPA) in the body of the HAVING clause :

• SELECT AVG(s.gpa), e.cid FROM enrolled AS e, student AS s

WHERE e.sid = s.sid

GROUP BY e.cid

HAVING AVG(s.gpa) > 3.9;

- **String Operations** : The SQL standard says that strings are case sensitive and single-quotes only.

• Pattern Matching: The LIKE keyword is used for string matching in predicates.

• “%” matches any substrings (including empty).

• “ ” matches any one character.

• Concatenation: Two vertical bars (“||”) will concatenate two or more strings together into a single string.

• String Functions : SUBSTRING(S, B, E) and UPPER(S).

- **Date and Time** : Operations to manipulate DATE and TIME attributes. Can be used in either output or predicates. The specific syntax for date and time operations varies wildly across systems.

- **Output Redirection** : Instead of having the result a query returned to the client (e.g., terminal), you can tell the DBMS to store the results into another table. You can then access this data in subsequent queries.

• New Table: Store the output of the query into a new (permanent) table.

SELECT DISTINCT cid INTO CourseIds FROM enrolled;

• Existing Table: Store the output of the query into a table that already exists in the database. The target table must have the same number of columns with the same types as the target table, but the names of the columns in the output query do not have to match.

INSERT INTO CourseIds (SELECT DISTINCT cid FROM enrolled);

- **Output Control**

• Since results SQL are unordered, we must use the ORDER BY clause to impose a sort on tuples:

SELECT sid, grade FROM enrolled

WHERE cid = '15-721' ORDER BY grade;

• The default sort order is ascending (ASC). We can manually specify DESC to reverse the order:

SELECT sid, grade FROM enrolled

WHERE cid = '15-721' ORDER BY grade DESC;

• We can use multiple ORDER BY clauses to break ties or do more complex sorting:

SELECT sid, grade FROM enrolled WHERE cid = '15-721' ORDER BY grade DESC, sid ASC;

• We can also use any arbitrary expression in the ORDER BY clause:

SELECT sid FROM enrolled WHERE cid = '15-721' ORDER BY UPPER(grade) DESC, sid + 1 ASC;

• By default, the DBMS will return all of the tuples produced by the query. We can use the LIMIT clause to restrict the number of result tuples:

SELECT sid, name FROM student WHERE login LIKE '%@cs' LIMIT 10;

• We can also provide an offset to return a range in the results:

SELECT sid, name FROM student WHERE login LIKE '%@cs' LIMIT 10 OFFSET 20;

**- Nested Queries**

• the inner query can access attributes from outer query .

1. SELECT Output Targets: SELECT (SELECT 1) AS one FROM student;

2. FROM Clause: SELECT name FROM student AS s, (SELECT sid FROM enrolled) AS e WHERE s.sid = e.sid;

3. WHERE Clause: SELECT name FROM student WHERE sid IN ( SELECT sid FROM enrolled )

• Example: Get the names of students that are enrolled in ‘15-445’

SELECT name FROM student WHERE sid IN ( SELECT sid FROM enrolled WHERE cid = '15-445' );

Nested Query Results Expressions:

• ALL: Must satisfy expression for all rows in sub-query.

• ANY: Must satisfy expression for at least one row in sub-query.

• IN: Equivalent to =ANY(). • EXISTS: At least one row is returned.

- **Window Functions :** Performs “sliding” calculation across a set of tuples that are related.

• Functions: The window function can be any of the aggregation functions that we discussed above. There are also also special window functions:

1. ROW NUMBER: The number of the current row. 2. RANK: The order position of the current row.

• Grouping: The OVER clause specifies how to group together tuples when computing the window function. Use PARTITION BY to specify group :

SELECT cid, sid, ROW\_NUMBER() OVER (PARTITION BY cid) FROM enrolled ORDER BY cid;

• We can also put an ORDER BY within OVER to ensure a deterministic ordering of results even if database changes internally :

SELECT \*, ROW\_NUMBER() OVER (ORDER BY cid) FROM enrolled ORDER BY cid;