**SQL Topics to know**

* **RDBMS** 🡪 Rational Database Management System
  + **Table** 🡪 has a set number of columns, which have data types and names. Can have any number of rows. We organize tables into schemas (almost like a name space). All these things live inside a database.
  + **Column** 🡪 a column is a vertical entity in a table that contains information associated with a specific filed in a table
  + **Row** 🡪 a record AKA a row of data in each individual entry that exists in a table
  + **Database** 🡪 contains tables and the tables schema.
  + **Schema** 🡪 similar to namespace. How we identify each table.
* **Normalization** 🡪 is the process of efficiently organizing data in a database. There are several reasons why we do this
  + Reducing redundancy
  + Improving consistency
  + Improving readability and performance
  + Improving referential integrity
  + Improving flexibility to future changes
  + 1NF (first normal form) 🡪 Requires:
    - A primary key (every row must be unique)
    - Atomic values and no repeating groups in a columns (don’t repeat your columns either)
  + 2NF (second normal form) 🡪 Requires:
    - Everything in 1NF
    - Everything fully depends on all candidate keys (no partial dependencies on any candidate key
  + 3NF (third normal form) 🡪 Requires:
    - Everything in 1NF and 2NF
    - No non-candidate key column can depend on any other non-candidate key column (no indirect dependencies)
* **Anomalies** 🡪 structural properties of a database or its components that can cause problems or lead to mistakes
  + **Insertion Anomaly** 🡪 the nature of a database may be used that it is not possible to add required piece of data unless another piece of unavailable data is also added.

**Example:** a library database that cannot store the details of a new member until the member has taken out of a book.

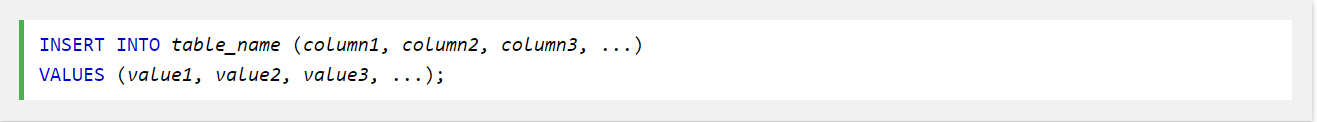
* + **Deletion Anomaly** 🡪 a record of data can legitimately be deleted from a database, and the deletion can result in the deletion of the only instance of other, required data.

**Example:** deleting a book loan from the library member can remove all details of the particular book from the database such as the author, book title, etc.

* **Modification/ Updating Anomaly (redundancy**) 🡪 a record of data redundantly stored and is functionally dependent on another record in the same table. If the independent record needs to be updated, then all instances of the dependent record need to also be changed, which could cause mistakes.

**Example:** updating the name of a client would require that we also update all dependent information about that client such a phone number, address, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Color** | **Supplier** | **Phone Number** |
| Widget | Blue | Contoso | 555-1234 |
| Thingybob | Blue, Red | Northwind | 555-4321 |
| Knicknack | Red, Black | Contoso | 555-1234 |

* Anomalies in the example table above
  + The values stored in color are not atomic
    - They are not indivisible (because the color field holds more than one value)
  + Redundancy (updation anomaly) 🡪 the phone number is carried with the supplier.
    - This is a problem because we have a repeat of the supplier “Contoso”
    - If the supplier changed their phone number, would have to update every instance of their phone number, which could lead to mistakes.
  + Deletion anomaly 🡪 we could lose some unrelated data if you wanted to delete unnecessary data.
    - Example: if we stopped selling Thingybob and deleted the row, we would lose the phone number for Northwind
  + Insertion anomaly 🡪 we can’t add new data without other data
    - Example: if we got information for a new supplier, we could not add that data without also selling one of their products.
* **Candidate Key** 🡪 minimal set of columns in a table that every other column depends on
* **Primary Key** 🡪 one of the candidate keys chosen to identify the rows in practice
  + Convention: Primary Key is usually underline, as is **Name** in the table above.
* **Foreign Key:** a set of columns which references another table’s primary key
  + Helps uphold referential integrity
* **Composite key** 🡪 when more than one column are required to define a key (candidate key, primary key, foreign key)
* **Referential Integrity (RI) 🡪** Relationships must always be consistent. In other words, any foreign key field must agree with the primary key that is reference by the foreign key.
* **Functional Dependency 🡪** Y on X mean “Y is a fact about X”
  + “for each possible set of X values, there is only one possible set of Y values”
  + One attribute is functionally dependent on another attribute if the value of the second attribute determines the value of the first. If you know the value of the second attribute, you can determine the value of the first.
* **DML (Database Manipulation Language)** 🡪DML refers to the subset of SQL that accesses and modified *data*(rows)
  + **Commands:**
    - **SELECT** 🡪 strictly speaking, is part of DQL (Doctrine Query Language) not DML but it is so closely related that SELECT statement is often considered part of DML.
    - Properties:
      * Used to access rows of data
      * Retrieved data stored in a result table, called **result-set**
    - There are SIX total clauses (including SELECT) that you can use
    - **FROM** 🡪 besides SELECT, FROM is the only other required clause to run the SELECT statement
      * Indicates table from which to retrieve data
    - **WHERE** 🡪used to filter records. Here, you indicate which conditions you want include (or exclude, depending on how you look at it)
      * **NOTE:** WHERE clause cannot use aggregate functions
      * **NOTE:** execution before GROUP BY clause, so you cannot filter by groupings using WHERE clause (see HAVING clause)
      * **Operators in the WHERE clause**
      * =
      * Equal to
      * <> or !=
      * Not Equal to
      * >, <, >=, <=
      * Greater than, Less than, Greater than or Equal to, and Less than or Equal to
      * BETWEEN
      * Between a certain range
      * LIKE
      * Search for a pattern
      * Often uses related operators
        + % and \_ are ‘wildcards’
      * IN 🡪 Specify multiple possible values in a column to look for
    - **GROUP BY** 🡪 the GROUP BY statement is often used with aggregate functions to group the result-set by one or more columns
      * **Aggregate Functions:** 
        + **COUNT, MAX, MIN, SUM, AVG**
    - **HAVING 🡪** specifies that a SELECT statement should only return rows where aggregate values meet the specified conditions
      * **NOTE:** Executes after GROUP BY clause
    - **ORDER BY 🡪** specifies how to sort selected data
      * **ASC =** ascending (default)
      * **DESC =** descending
      * **Executes last BUT before SELECT**
  + **OTHER COMMANDS**
    - **INSERT INTO** 🡪 used to add data (rows) into a table
      * Two ways to write INSERT INTO statement
        + Specify column names before listing values (best practice for readability)
        + 
        + Omit column names and list only values. Must provide entry for each field in table
      * Can create temporary tables with #
    - **UPDATE** 🡪 used to modify existing records in a table
    - **DELETE** 🡪 used to DELETE existing records of table
      * Used for deleting rows in a table, not whole tables
    - **TRUNCATE TABLE** 🡪 used to deleting all rows in a table.
      * Does not delete the table itself, only the rows (see DROP)
* **DDL (Data Definition Language)** 🡪 DDL refers to the subset of SQL that is used to define columns and tables (access the structure of our data). DDL does not have access to the data (rows) themselves
  + **Commands:**
    - **CREATE 🡪** Used to build a new structure (database, schema, table, view etc). Example: CREATE DATABASE
      * When creating a table, fields must be specified.
        + **NOTE** 🡪 fields are nullable by default, but is good practice to be explicit (readability)
      * **Views** 🡪 views are not tables, but you can think of them as read-only tables.
    - **ALTER TABLE** 🡪 used to add, delete, or modify fields in a table.
      * Can add or drop constraints to a field with this command.
        + (NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT, INDEX)
      * ON DELETE CASCADE 🡪 use the ON DELETE CASCADE option if you want rows deleted in the child table when corresponding rows are deleted in the parent table.
      * ON UPDATE CASCADE 🡪 like “ON DELETE CASCADE” this keyword specifies that you want all referenced tables to be updated when you update the parent table
      * Can also add fields with ALTER TABLE statement
    - **DROP** 🡪 the inverse operation of CREATE. Can DROP (DATABASE, SCHEMA, TABLE, VIEW ETC.)
* **User-defined functions**:
  + Use CREATE FUNCTION or ALTER FUNCTION
  + Can be used to return a whole table (in result-set)
  + Functions can be used in the SELECT clause
  + NOTE: to functions, data, and structure of database is read-only
    - Functions cannot create, modify, add, insert, drop, or otherwise manipulate data
    - For this, use STORED PROCEDURES
* **Stored procedures**
  + Used to change data or database structure
  + Cannot be used in SELECT clause
* **Transactions** 🡪 is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased. a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.
  + **ACID:**
    - **Atomicity** 🡪 in a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are.
      * **Why is this important:**
    - **Consistency** 🡪a transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.
    - **Isolation** 🡪 a transaction in process and not yet committed must remain isolated from any other transaction
    - **Durability** 🡪 committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.
  + **Isolation Levels**
    - Transactions specify an isolation level that defines the degree to which one transaction must be isolated from resource or data modifications made by other transactions. Isolation levels are described in terns of which concurrency side effects, such as dirty reads or phantom reads, are allowed.
      * **Dirty Read** 🡪 a dirty read (uncommitted dependency) occurs when a transaction is allowed to read data from a arrow that has been modified by another running transaction and not yet committed.
      * **Non-repeatable Read** 🡪 a non-repeatable read occurs, when during the course of transaction, a row is retrieved twice and the values within the row differ between reads.
      * **Phantom Read 🡪** a phantom read occurs when, in the course of a transaction, new rows are added by another transaction to the records being read
    - The trade-off for isolation levels is that better protecting isolation levels are lower performing
* **Triggers** 🡪 a database trigger is special stored procedure that is run when specific actions occur within a database
* **Multiplicity** 🡪 the Multiplicity attribute of a relationship specifies the cardinality or number of instances of an EntityType that can be associated with the instances of another EntityType
  + **One-to-One 🡪** make one item a column (unique) in the table of the other. You can have FK back and forth, but one must be nullable.
  + **One-to-Many** 🡪 a foreign key from B(many) to A(one)
  + **Many-to-Many** 🡪 junction table. Multiple keys