SQL

Scalar and Aggregate Functions

- scalar: single input, single output
    - round, trim, substr, length, upper, lower, concat
- aggregate: group of inputs, single output
    - count, avg, sum, max, min
    - GROUP BY: used with aggregate functions
    - HAVING: used with aggregate functions

```
SELECT col1, count(col2) FROM table GROUP BY col1;
SELECT count(col1) FROM table GROUP BY col1 HAVING col2 = value;
```

General SQL Review Cont.

- Relational Database
    - a database where we store data as it relates to other data, or as connections to other data. follows the relational model created by E.F. Codd.
    - SQL: a language used to query, modify, and define a relational DB.
    - table: entity comprised of columns and rows, sometimes referred to as "relations"
        - columns are named - same type of information. these would correspond with fields of a class in Java (sometimes they are actually referred to as fields)
        - rows are data entries - "complete" sets of information. these would correspond with instantiated objects in Java (they are sometimes referred to as records)
    - primary key: the unique identifier of a row
        - natural key: a key arising from the dataset; something that is naturally a unique identifier
        - candidate key: a subset of columns that COULD be a primary key; can be multiple columns, must be unique & non-nullable
        - composite key: a key comprised of multiple columns
    - candidate key conditions
        - no sensitive information (e.g. SSN, password, etc)
        - non-volatile (i.e. can't be changed) (e.g. username, email)
            - could cause orphaned fields in foreign keys
    - surrogate key
        - generated by the database then associated with the data
        - basically the opposite of a natural key, this is an arbitrary value whose sole purpose is to act as the primary key (usually an ID)
    - best practice regarding keys is sometimes debated, but:
        - now that memory is less of a concern, surrogate keys are much safer and more practical than natural keys

        - composite keys are complicated to work with, especially when connecting to Java, and even moreso when working with ORM frameworks like Hibernate and SpringData which are commonly used today, so they are discouraged
- SQL Sublanguages
    - sometimes these are debated as well - numbers vary from 3-5
    - if three: DDL, DML, DCL
    - if five: DDL, DML, DQL, DCL, TCL
    - we typically consider there to be 5, but it is important to know that some interviewers do not feel this way and there are valid reasons for other options ("don't die on this hill")
    - DDL: data definition language
        - this is used to define tables, constraints, views, etc.
        - CREATE, ALTER, DROP, TRUNCATE
    - DML: data manipulation language
        - this is used to manipulate rows in tables
        - in the "three sublanguages" view, DQL is included here so as to include all CRUD operations
        - INSERT, UPDATE, DELETE
    - DQL: data query language
        - this is used to query the data
        - in the "five sublanguages" view, this is a separate sublanguage because it can get very complex; there are many things that can get involved in querying data
        - SELECT
    - DCL: data control language
        - this is used to manage permissions of database users
        - GRANT, REVOKE
    - TCL: transaction control language
        - in the "three sublanguages" view as well as sometimes in a "four sublanguages" view, this is included with DML as well
        - this manages transactions, aka groups of DML statements that are related to each other
        - ROLLBACK, COMMIT, SAVEPOINT, BEGIN
- PostgreSQL
    - open source DBMS
    - reliable with good performance, cheaper alternative to Oracle SQL
    - is not case sensitive (all SQL dialects are case insensitive)
        - in general, you will see keywords in all caps and other words in lowercase
    - naming conventions are snake_case due to the lack of case sensitivity
        - this includes table names, column names, etc.
    - database: collection of objects for storing info
    - schema: structure/organization of data
        - separate schemas can be created to keep related data together
- SQL
    - foreign key: a subset of columns that reference the primary key of another table
        - this is how we build up our relationships in a relational database

        - example: a CAT table/relation for our CatApp would have the following columns/fields: ID, NAME, AGE, BREED_ID, STATUS_ID
            - BREED_ID and STATUS_ID are foreign keys to the BREED table and the STATUS table
    - basic queries
        - SELECT (columns) FROM (table)
        - WHERE: sets a condition, such as WHERE id = 2;
        - ORDER BY: allows you to choose which column to sort the result set by, such as ORDER BY name
    - data types to know: integer/int real decimal/numeric serial varchar(n) char(n) boolean/bool date time timestamp money
    - joins
        - joins are used to query data from multiple tables at the same time.
        - types of joins:
            - inner join: only returns the rows that have a match
            - left outer join: returns all rows from the left table, with nulls in the right-side rows that don't have matches.
            - right outer join: returns all rows from the right table, with nulls in the left-side rows that don't have matches.
            - full outer join: returns all rows from both tables with nulls in rows on either side where matches do not exist.
            - cross join: the Cartesian product of all of the rows from both tables; i.e. all possible combinations of rows from both tables. the result sets of these can obviously get very large, so only use when necessary.
            - self join: joining a table with itself in order to combine related data from the same table. a great example of this is when an EMPLOYEE table has a foreign key referencing that employee's manager who is also an employee and therefore also in the EMPLOYEE table.
        - examples: SELECT * FROM TABLE_A JOIN TABLE_B ON A = B; SELECT * FROM TABLE_A JOIN TABLE_B USING ID; SELECT * FROM TABLE_A LEFT OUTER JOIN TABLE_B ON A = B;
        - the JOIN keyword alone performs an inner join.
        - if we have to use JOINs anyway, what is the point of designating certain fields as foreign keys?
            - a foreign key is a type of constraint, meaning that it restricts certain DML from being executed if the result does not fulfill the constraint. if you try to add a row but the value for the foreign key column does not reference any row in the table that it is referencing, the database will not allow you to perform that insert. if you try to delete a row that is being referenced by a foreign key in another table, it will not let you perform that delete, and so on.

More SQL!

- constraints

    - constraints are rules set up for columns that allow the database to restrict certain actions
    - foreign key constraint (discussed above): uses the REFERENCES keyword. for composite keys, constraint is defined after the columns are defined:
        - FOREIGN KEY (first_id, second_id) REFERENCES other_table (first_id, second_id)
    - PRIMARY KEY constraint: for composite keys, constraint is defined after the columns are defined:
        - PRIMARY KEY(first_id, second_id)
    - UNIQUE constraint: does not allow duplicate values in this column; good for things like usernames
    - NOT NULL constraint: does not allow anything in the column to be null
    - CHECK constraint: allows you to set a condition for the column, such as CHECK(NUM > 10)
- aliases
    - aliases give things a temporary name for the purpose of queries or result sets.
    - table aliases allow complicated queries to be made more succinct and potentially easier to read
        - e.g. SELECT A.ID, A.NAME, B.NAME FROM TABLE_A AS A JOIN TABLE_B AS B ON A.ID = B.ID;
    - column aliases allow result sets to be more readable and more practical, and they also allow nested queries to be easier to read as well
        - e.g. SELECT A.NAME AS "Employee", B.NAME AS "Boss" FROM TABLE_A AS A JOIN TABLE_A AS B ON A.BOSS_ID = B.ID; multiplicity relationships
- one-to-one: your standard relationship where exactly one row in the first table references one row in another table
    - example: one user has one address, and that address is not shared by any other user
- one-to-many/many-to-one: exactly one row in one table is referenced by potentially many rows in another table
    - example: one cat only has one breed, but that breed may be applied to many cats. (same thing applies to status)
    - example: one cat only has one owner, but that owner may have many cats.
- many-to-many: several rows in one table can reference several rows in another table. usually a multiplicity table is created for this to allow it to be used more like two one-to-many relationships (easier to work with.)
    - example: one cat can have many special needs, AND those special needs can be related to many different cats. database normalization (overview)
- the purpose of database normalization is to decrease data redundancy and increase data integrity. it is part of E.F. Codd's relational model.
- database normalization makes your database drastically more maintainable, as small changes do not need to break everything - things can be inserted and removed fairly easily.
- there are guidelines for normalizing a database, and these go up in steps as the database becomes more normalized.
    - in most cases, the goal is to reach 3NF, or "third normal form."
- UNF: unnormalized form
    - primary keys, no repeating groups

- 1NF: first normal form
    - UNF + atomic columns (only one value in a cell)
- 2NF: second normal form
    - 1NF + no partial dependencies (fields must depend on the entirety of every candidate key)
- 3NF: third normal form
    - 2NF + no transitive dependencies (fields must only depend on candidate keys)

Order of Operations on select statements (Group by, having, etc.)

https://learnsql.com/blog/sql-order-of-operations/