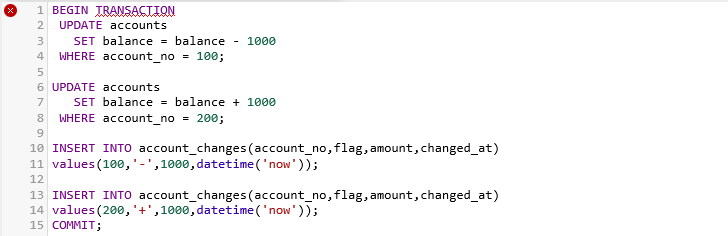
**Transactions**

* A transaction is a group of **one or more SQL statements executed together**, with the help of commands from the SQL Sublanguage Transaction Control Language (TCL).
  + TCL Commands:
    - **BEGIN** - Starts a transaction
    - **COMMIT** - Completes a transaction, save the queries to the database
    - **SAVEPOINT** - Sets up a temporary save while the transaction is running
    - **ROLLBACK –** Reverts back to the last savepoint in case of errors
* Transactions are important for data integrity, and they are commonly used when you need a group of statements to either succeed or fail together - no in-between.
* Transactions operate under a certain set of properties – the **ACID Properties**
* **A - Atomicity (atomic)** 
  + The entire transaction succeeds in one go, or it doesn’t happen at all.
  + Every SQL statement in the transaction must complete successfully, or we undo everything the previous statement did.
    - If 99 statements succeed, and the last one fails, all 99 get rolled back!
  + Atomic (Atomus - The transaction can’t be broken down further. We can’t have a smaller piece of the total transaction succeed alone)
* **C - Consistency (consistent)**
  + The database has constraints… (PK/FK, Unique, Not Null, etc.)
  + While transactions are being executed, we never enter a state where constraints are disregarded.
    - Can’t deviate from the constraints of your DDL while transactions are being executed. Transactions are not above the law.
* **I - Isolation (isolated)**
  + In practice, many transactions can be running concurrently (at the same time).
  + Each individual transaction runs as if it was the only transaction running. This avoids collision of the database.
  + Transactions can’t interfere with each other! The database implements this for us.
    - SQL has different transaction isolation levels that we can change - the more isolated transactions are, the less they can affect each other, and the more reliable they become
    - BUT the more isolated we make a transaction, the slower our database will function.
    - There are many different isolation levels! Look into it if you want (nobody will). The default for Postgres is “read committed”.
* **D - Durability (durable)**
  + When the transaction is complete, all data is saved to the database's physical medium.
  + I other words, the data won’t be changed or lost after a transaction is complete.

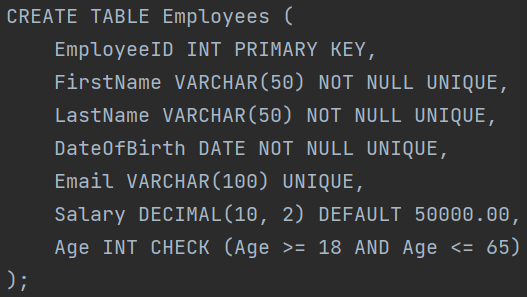
**\*Example transaction**

  
\*Note the BEGIN and COMMIT commands - part of TCL

**Advanced Schema Topics-------------**

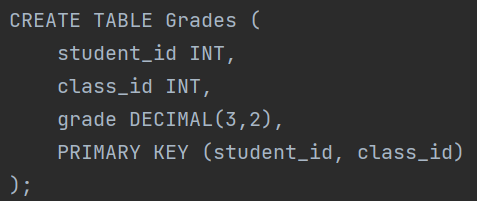
* Some common Constraints
  + **UNIQUE –** Does not allow two records to share the same value for a column.
  + **DEFAULT -** Defines a default value if one is not provided.
  + **NOTNULL -** This column can’t contain a null value.
  + **CHECK -** Checks a condition to determine if a value it’s allowed or not.

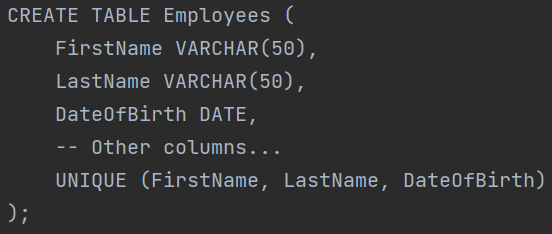
Here’s an example of a table using each of the constraints mentioned above:

  
Careful with constraints - can you foresee some problems with this combination of constraints?

**Alternative Primary Keys**

* **Composite Keys:**
  + Composite Keys are **unique identifiers made up of one or more columns**. Typically, they’re only used when a single primary key is not enough to uniquely identify a record.
    - The classic example is a table that stores class grades. A composite key can be made up of student\_id and class\_id to uniquely identify which student/class combination the grade belongs to.



* **Unique Keys:**
  + A unique key is **a set of one or more columns that are unique across the dataset.** This means no two records will ever have the same combination of these columns.
    - These are perhaps most easily accomplished with the UNIQUE constraint!
    - 
    - Now, every record must have a UNIQUE COMBINATION of these three columns.

**PL/SQL Topics--------------------**

**SQL Functions**

* SQL has several functions that let us perform more complex calculations on data.
  + <https://www.techonthenet.com/postgresql/functions/index.php>
* **Scalar functions**
  + **Take in a single value and return a single value** based on some calculation.
  + Examples include upper, lower, round, now (returns the current time)
    - 
* **Aggregate functions**
  + **Take in a group of values and return a single value** calculated from the set
  + Avg, min, max, sum, count (count’s the number of returned records)
    - 
* You can also create custom **user-defined functions** if there is some specific functionality you would like to define for ease of use. We won’t do this :)

**Stored Procedures**

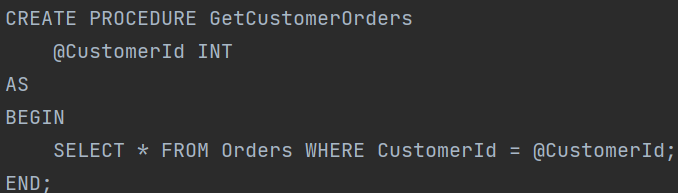
* A stored procedure is a named collection of SQL statements that can be **stored and executed on a database**. It can be called and executed multiple times by different applications or users.
  + What’s the point?
    - Stored procedures are typically used to **group frequently executed or complex database operations**. They provide several benefits, including improved performance (speed), code reusability, security, and abstraction of database operations.
    - Would you rather retype that complex 15-line query multiple times a day? Or store it once and call it at will?

**Real world example:**

Consider an e-commerce application where you need to perform a complex operation like placing an order. This operation might involve several steps:

1. Decrease the stock of the ordered product.
2. Create a new order record.
3. Create order detail records for each product in the order.
4. Update the customer's total amount spent.
5. Probably other stuff.

Each of these operations requires a database operation! Why not lump them into one larger procedure? Again, this improves speed, reusability, security, and operation abstraction.

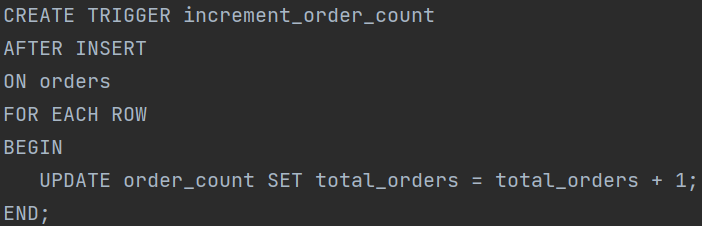




Here’s a very basic example of the syntax, but stored procedures are typically way more complex!

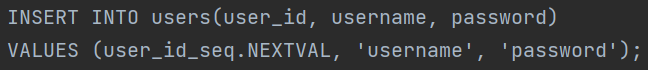
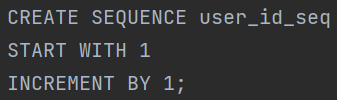
**Triggers**

* SQL Triggers are special stored procedures defined to execute automatically in response to certain events. In other words, they’re **procedures that get triggered by certain events in the database.**
  + These events can be insertions, updates, or deletions made on a table.
  + Triggers are most often used to maintain data integrity, perform secondary updates on existing data, or to replicate data.

  
In this example, we want to increase the value of ”total\_orders” in the ”order\_count” table by 1 every time a new order is inserted into the database. Imagine processing millions of orders a day... Having a trigger automatically perform the update saves a LOT of time and processing power.

**Sequences**

* SQL sequences are database objects that **generate a sequence of unique integers**. They are often used to automatically generate primary key values, to ensure that each record in a table has a unique identifier.



“But BEN!” You may ask... “We usually auto generate our primary keys! What’s the point of using sequences?”

* You’re right, you probably won’t need to use these. One advantage is control, with the ability to **choose the starting number, and increment by more than 1**.
* Another advantage is **portability.** Sequences are part of the SQL standard while serial PKs are Postgres specific.
  + Basically, Sequences are seamlessly portable between different SQL dialects while the serial datatype is not.

------------------------------------------------------------------------

**DBeaver Quality of Life Stuff**

**-Dark Mode: window => preferences => user interface => click on appearance => theme**

**-Show Line Numbers: right click next to the cursor in a script => show line numbers**

**-Capitalize Keywords: Window => Preferences => Editors => SQL Editor => Formatting => Keyword Case (change to Upper).**