An overview of the tables that will occupy the “laid out graph” SQLite database to be outputted by collate_clusters.py, with accompanying regular expressions specifying the type of input for each field.

(Note that “NUM” just indicates whatever we consider a number in xdot2cy.js – some pretty strange numbers can arise in .xdot files and in JavaScript, e.g. using scientific notation, so to avoid writing the same thing out here we just reference the same number-detection regex.)

TABLE contigs

Contig ID	Len (bp)	DNA Seq (fwd)	DNA Seq (rev)	Depth	Cmp. Rank	x	y	w	h	Shape	Parent Cluster ID
c?\d+	\d+	[ATCG]+	[ATCG]+	\d+	\d+	NUM	NUM	NUM	NUM	(house|invhouse)	((B|R|C|Y)c?\d+(_c?\d+)*)|NULL

A Parent Cluster ID value of NULL just indicates that this contig is not located within a cluster.

TABLE edges

Source Contig ID	Target Contig ID	Multiplicity	Cmp. Rank	Ctrl Pt String	Ctrl Pt Ct	Parent Cluster ID
c?\d+	c?\d+	\d+	\d+	(NUM NUM)+	NUM	((B|R|C|Y)c?\d+(_c?\d+)*)|NULL

Note that we don't associate edges with clusters when creating the initial DOT file (.gv), since it doesn't really matter there. However, “interior” edges of a cluster are associated with (i.e. nested within the declaration of) clusters in the resulting .xdot file, and we use an edge's cluster for a few things in the Javascript viewer. Hence why edges here have a Parent Cluster ID value (as with nodes' Parent Cluster ID value, NULL just indicates that this edge isn't inside a cluster).

Also note that we wait to parse + convert control points to Cytoscape.js' coordinate system and relative (instead of absolute)-based format, in order to offload the work of viewing to xdot2cy.js.

TABLE clusters

Cluster ID
(B|R|C|Y)c?\d+(_c?\d+)*

I was thinking of having another field in this table for “Cluster type” (the possible values for which would be just the union of all one-letter cluster types), but we can get the same information from just the first letter of any cluster's ID. This is more space-efficient, arguably more elegant, and less error-prone (we only have to update one value).

See the discussion below for components re: how to record member elements. I decided to do the same thing here—to give each cluster's member nodes/edges a

reference to this cluster ID to easily facilitate selecting all nodes/edges with a given cluster ID, instead of trying to create a hack-ish “array” of member elements.

Also, we don't need to store position information for clusters because they autofit to the minimum size needed to cover their child nodes.

TABLE components

Component Size Rank (1..n)	Contig Count	Edge Count	Total Contig Length (bp)	Bounding Box x	Bounding Box y
\d+	\d+	\d+	\d+	NUM	NUM

I was going to have a list of contigs making up the component attached to the definition in the database for each component, but it turns out SQLite doesn't actually have built-in support for lists/arrays.
We could still relatively easily just store the list of contigs as a space-separated string or something (e.g. “1 c1 60 c60...” or something) and then split that to parse it in the Javascript viewer, but that sounds like a bit too much work (and a bit too error-prone). So what I'm doing instead (I guess it's the more “elegant” way, anyway, even if it is a bit less efficient) for now is just giving each contig and edge a “component” field that refers to the size rank of the component of which that contig is a member, and from there in the Javascript side of things we can do something like
`SELECT * FROM contigs WHERE COMPONENT_RANK = n` to get all the contigs comprising the `n`th biggest component in the graph, and
`SELECT * FROM edges WHERE COMPONENT_RANK = n` to get all the edges connected to the contigs in that component.
Also, note that (since we split up connected components into their own graph layouts) each connected component is really its own “graph,” which is why we store bounding box information/etc on the component level.

TABLE assembly

Graph filetype	Contig Count	Edge Count	Connected Component Count	Total length (bp)	N50 (bp)
(LastGraph|GraphML)	\d+	\d+	\d+	\d+	\d+

Note that the filetype field is really important—we'll need to use it to adjust what sorts of features are available in the graph viewer, since GML files carry inherently less information (e.g. no DNA sequences) than LastGraph files do. (As we support more types of assembly graph files, we'll add more possible values to the regex here.)
Also: for now, let's include RC nodes and edges in the total count/total length/etc, to err on the side of accuracy. If/when we implement the single graph thing for LastGraph/etc files, we can just halve those numbers, I guess.

We can definitely add more interesting statistics here; this is just an initial sampling of some important ones. Although all of these should be fairly simple to calculate, I would prefer to calculate all of them in collate_clusters.py, to separate the graph preparation (pattern detection, splitting by components, laying out) and viewing (via Cytoscape.js) portions of AsmViz.