Modeling Power: Data Models and the Production of Social Inequality

by

Nikki Lane Stevens

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved March 2022 by the
Graduate Supervisory Committee:

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Software systems can exacerbate and cause contemporary social inequities. As such, scholars and activists have scrutinized sociotechnical systems like those used in facial recognition technology or predictive policing using the frameworks of algorithmic bias and dataset bias. However, these conversations are incomplete without study of data models: the structural, epistemological, and technical frameworks that shape data. In *Modeling Power: Data Models and the Production of Social Inequality*, I elucidate the connections between relational data modeling techniques and manifestations of systems of power in the United States, specifically white supremacy and cisgender normativity. This project has three distinct parts.

First, I historicize early publications by E. F. Codd, Peter Chen, Miles Smith & Diane Smith, and J. R. Abrial to demonstrate that now-taken-for-granted data modeling techniques were products of their social and technical moments and, as such, reinforced dominant systems of power. I further connect database reification techniques to contemporary racial analyses of reification via the work of Cheryl Harris.

Second, I reverse engineer Android applications (with Jadx and apktool) to uncover the relational data models within. I analyze DAO annotations, create entity-relationship diagrams, and then examine those resultant models, again linking them back to systems of race and gender power. I craft a method for performing a reverse engineering investigation within a specific sociotechnical context -- a situated analysis of the contextual epistemological frames embedded within relational paradigms.

Finally, I develop a relational data model that integrates insights from the project's reverse and historical engineering phases. In my speculative engineering process, I suggest that the temporality of modern digital computing is incommensurate with the temporality of modern transgender lives. Following this, I speculate and build a trans-inclusive data model that demonstrates uses of reification to actively subvert systems of racialized and gendered power. By promoting aspects of social identity to first-order objects within a data model, I show that additional "intellectual manageability" is possible through reification.

Through each part, I argue that contemporary approaches to the social impacts of software systems incomplete without data models. Data models structure algorithmic opportunities. As algorithms continue to reinforce systems of inequality, data models provide opportunities for intervention and subversion.

ACKNOWLEDGMENTS

This project, like all major projects, was a community and collaborative effort. I have been keeping a list of people to mention in my acknowledgements since before I began this project, afraid that I would forget people's small generosities, and thus give the impression that any part of this was a singular endeavor. As a result, these acknowledgements may be far too long by academic standards; they are also far too short.

I was fortunate enough to proceed through the fraught experience of graduate school with a cohort of powerful people. My experience would have been much less fun without the companionship of Sean McCafferty, Noa Bruhis, Taylor Genovese, Shivam Zaveri, Anna Muldoon, Vanya Bisht, Martín Perez Comisso, and Amanda Arnold. Our weekly meetups at Casey's kept me grounded during that first difficult year. While at ASU, I also had the opportunity to spend time as a Graduate Fellow at the Institute for Humanities Research. At the IHR, I got to share space with Leah Newsome, Stephanie Silva, Liz Grumbach, and Dr. Mara Godoy-Lopez. Mara, in particular, has been a gift of support, care and encouragement.   While in Phoenix, I coached and performed at The Torch improv theater, a delightful distraction from the business of school. During that time, I was also able to reunite with Rachel Land Nystrom and Eric Nystrom – a surprise time travel reunion with two cherished old friends.

Part of this work was completed at Dartmouth College, while working in Jacque Wernimont's Digital Justice Lab. At Dartmouth, I was supported by Dan Chamberlain, Anna Gallert, Brinker Ferguson, Allie Martin, and so many lovely Dartmouth undergrads, especially Kirby Phares and Caroline Casey. In Utah, where I landed during the first lockdown, I thank Cale Carthey, Ruth Armbruster, Lou Schafer, the Iowa House community, and all of the perfect and beautiful queer and trans baristas at Sugar House Coffee, where I did much writing.

As a first-generation graduate student, I have been so aided by those senior to me helping me navigate the intricacies of institutions, conferences, budgets, job markets, and all the hidden curriculums. Sarah Florini and Anna Lauren Hoffmann have been especially generous here.

The pandemic began shortly after I began writing, and when it started, I began remotely coworking with some new and old friends. That group, now affectionally called Good Zoom, is still going after more than 2 years and we have seen each other through graduations, marriages, house purchases, the additions and losses of pets, big moves and scary transitions. I am so grateful to my Good Zoom friends: Molly Morin, Liz Dietz, Madison Weist, Allie Martin, Els Woudstra, Esther Grace Witte, Allie Tova Hirsch, Kirsten Lee, Cassius Adair, and Jackie Pogue. Also on Zoom, I was greatly supported by writing groups, including members of Good Zoom plus, variously, Jenny Brian, Dagmar van Engen, and Nadja Eisenberg-Guyot.

As I began this dissertation, I lost Poodle, the old dog I mention in the introduction. Poodle The Bulldog was my constant porcine companion from 2009 - 2020, and an eager accomplice to early research on pet surveillance that informed this project. Since her death, I gained two dogs: Marty and Chicken, who have been stalwart and delightful companions, despite their disinterest in my work. Notable associate pets include Donut Pepper, Sydney McNab, and Emma The Cat.

My committee members – Jacqueline Wernimont, Katina Michael, Jennifer Richter, and Marisa Duarte – were steadfast supporters and cheerleaders from our very first convening. Marisa is wise and kind and, when I was absolutely devastated by the scope of trans anguish in the archives and in the scholarship, helped me understand the potency of those archives. Marisa's work has underscored the importance of continuing to think about those to whom I'm accountable. Jen has been encouraging, honest, pragmatic, and supportive. I would not have ended up at ASU (nor stayed during a rough first semester) were it not for Jen's candor. Katina's enthusiasm for her work and her students is contagious. She is a vibrant and encouraging thinker and leader who has a talent for connecting people and ideas to each other. Jacque was both my chair and advisor and in those roles was an absolute force. She was generous with her time and thinking, giving me a true apprenticeship for learning how to think in a very particular way. Any critical skill I developed during school is in no small part a credit to her generosity and patience that were strong enough to counter my stubbornness. She also fed me frequently, cared for me in the wake of a very sudden surgery, and has made a grave for Poodle in her backyard (as an incomplete list). It is with immense gratitude that I recount the opportunity to study with and get to know such fine scholars.

Finally, everything is more possible with the support and care of those closest to me. Mikel Wadewitz has been there for over a decade and is unwaveringly kind and supportive. Mikel, Sarah Florini, Rowan Q. Buckton, and Liz Dietz have heard so much of this project, both in and out of context, and have supported me through rounds of despair, enthusiasm, bravado, and exhaustion. I received extra care from my professional collaborator turned wife Molly Morin. I strongly recommend falling in love with and moving in with and then getting engaged to the best artist and one of the kindest people you know during dissertations. 10/10, would marry again.

TABLE OF CONTENTS

APPENDIX Page

LIST OF FIGURES

INTRODUCTION

A few years ago, I planned to spend my December holiday break watching movies. The weather was terrible, driving conditions a little treacherous, and my aging dog was having an increasingly hard time in the snow and the cold. After the end of the semester, the dog and I settled into the couch together and I logged in to Netflix to find a movie. Among other holiday recommendations, Netflix suggested I watch *A Madea Christmas*. It had been years since I had seen a Tyler Perry movie; and, after watching *A Madea Christmas*, I spent the whole break idly watching all the other movies in the *Madea* franchise.

It is by now a joke that once Netflix "thinks you're Black" (i.e., you watch several movies with majority-Black casts) it begins surfacing other movies with all-Black casts. For months after the break, Netflix did just this, recommending an immense amount of Black content. This was partially a delight—I had forgotten how many movies Queen Latifah has been in—and partially a curiosity. Where had these movies been? I watched the algorithm try to reconcile my *Madea* marathon (and subsequent Queen Latifah binge) with other escapist mid-semester viewing habits, like *Star Trek: The Next Generation*. Prior to this, Netflix did not recommend these movies to me. Was it my whiteness, or the fact that I mostly use Netflix as background noise filled with childhood science-fiction nostalgia? Netflix asserts that this recommendation algorithm is not based on the racial or ethnic demographics of its users—it claims to not track that data. Indeed, I suspect the initial *Madea* recommendation was based on its holiday theme, rather than the Black cast.

On the surface, it is not a problem to recommend movies like the ones people have already watched. (What exactly makes *Set it Off* similar to *Madea's Family Reunion*?) As many people have pointed out, this behavior is problematic not because it serves Black content to viewers who have already watched Black content, but because it *does not* serve that content to viewers who have not. This algorithmic result reinforces "white movies" as normative and popular, and "Black movies" as niche interests that only warrant recommendation once someone has indicated a desire. However, Netflix claims no ill-intent, only that its machine learning algorithms are designed to foster interaction: to get users to click on thumbnails, read synopses and ultimately view content. The machine learning algorithm is only reproducing its inputs—people who have watched Tyler Perry movies have also watched Queen Latifah movies. It is simply the algorithm.

But Netflix, like all software, is more than just the complicated sets of instructions we call algorithms. Netflix also has immense amounts of data, huge databases, and data storage, all connected through complex workflows and managed both by automated systems and individual people. There are many opportunities for biased, unjust, or otherwise problematic outcomes to manifest. In this project, I focus on what Paul Dourish calls "algorithmic others" -- the other parts of software that contribute to its form and functionality (Dourish, 2016). Of these algorithmic others, I focus even more specifically on data models. Data models are sets of design documents that, among other things, determine how a database is constructed and how data moves through a software system.

The options available on forms (particularly web forms) are a common way that individuals may experience a software's data model firsthand. Consider the banal act of filling out a registration form on a website.[1] The ways that website forms request data is typically linked to the site's underlying data models. In other words, the web forms request data in a format to match what the data model is expecting. When the data model includes

[1] Discussing whether or not a website counts as "software" is out of scope here. For my purposes, it absolutely does.

fields for a first, middle and last name, those fields are echoed in the first, middle and last name fields of the form.

The creation of first, middle and last name fields reveal the biases and assumptions of the data model designers. Restrictions on the length of a first or last name (both minimums and maximums) entered into fields are racist manifestations of US-based expectations for Anglo-Saxon names (Wachter-Boettcher, 2017). When a web form's field length restricts a last name field or provides only an Anglocentric "first, middle, last" configuration it is typically because there are database fields expecting those three values (and only those three values). Those database fields are in turn created in response to data models (i.e., high-level database design documents) designed to track all the information that the system needs to store. However, unlike other aspects of computing (like memory storage limitations or even the foundational binary computing language), data models are flexible and very often a "blank slate" opportunity for software development teams to model the aspects of the world they wish to monitor and, within racial capitalism, monetize.

This dissertation takes data models as its focus, and throughout I offer my main argument: that data models are important social and cultural artifacts worthy of scholarly study. Furthermore, when we look at data models, we will find that they support existing systems of power. Data models are also crucial sites for intervention in the unjust or undesirable outcomes of software systems. I approach data models in several ways. First, I locate them within the process of software production, demonstrating their foundational role in the built software product. Second, I critically analyze early relational data modeling texts, demonstrating how techniques and epistemologies of data modeling support systems of power. In addition to studying the techniques of data modeling, I use reverse engineering to peel back the layers of software and demonstrate how to examine data models within published commercial software. Finally, I use data models as a form of intervention in software systems, using techniques of relational data modeling to both explore ways of subverting power and to make transparent the ways that data models reinforce existing power structures. Throughout, I focus on the importance of data models to our understanding of software.

Theoretical Frameworks

Science and Technology studies (STS) is a foundationally interdisciplinary field. My work leverages that interdisciplinarity and combines scholarship from STS, critical data and critical algorithm studies, surveillance studies, infrastructure studies, whiteness studies and transgender studies. I discuss each below.

I hypothesize that within the design of data models, we will find that data structures match the designers' visions both of the world they are co-creating and the users they are courting. While there is voluminous scholarship about algorithmic matters, there is relatively less so on the oppressive potential of data models.[2] I pull together elements of the fields mentioned above to craft an urgently needed project—a technosocial investigation into the ways that data models can both support structural power and be used to subvert it.

Science and Technology Studies

Science and Technology Studies scholarship has long acknowledged that technological systems are political, value-laden affairs (Winner, 1986). The values represented are those of the creators, funders, and engineers; and, under the social construction of technology model, society itself (Pinch & Bijker, 1984). While software

[2] For more on algorithmic ethics see Mittelstadt et al. (2016), algorithmic regulation: Lodge & Mennicken (2017); algorithmic violence: Owen (2015), algorithmic auditing: Sandvig et al. (2014).

engineering communities do not always consider the "politics" of their artifacts, conversations about the human impacts of computing technology are nearly as old as the technology itself.[3] Recently, the software engineering industry has been receptive to scholarship about the social impacts of their products and the ways that technology can create injustice and produce disadvantages for groups of people. Work documenting the unjust outcomes of technologies like predictive policing (Ferguson, 2017), facial recognition systems (Gates, 2014), and housing algorithms (Eubanks, 2018) has encouraged corporations, and engineering and academic communities to investigate solutions.

While these solutions could take many forms—infrastructural, social—oftentimes the solutions are technical, and specifically algorithmic. For example, data scientists can train models to react "fairly" using "Variational Fair Autoencoder (VFAE). VFAE characterizes 'fairness' as a representation that is explicitly invariant with respect to some known aspect of the dataset" (Xu, 2019). The use of algorithmic interventions to correct for "unfairness" centers the disadvantage experienced by individuals and sidesteps the structures and systems that produce dis/advantages. As Anna Lauren Hoffman notes, interventions to correct for disadvantage (to make systems "fair") "limit us to solutions that are, at best, reactive and superficial" and occlude the underlying structural logics that are foundational to the dis/advantages (Hoffmann, 2019, p. 904).

In a world in which increasing numbers of critical systems are automated, a focus on the systems and operations controlling that automation makes sense. Importantly, AI has received immense focus in part because the algorithms within AI systems seem inscrutable. But AI does not come from nowhere—systems are trained on large datasets—datasets that

[3] See Baldwin & Shultz (1955) and Hoos (1960) for early business examples and Dow (1949) for an early education example.

are simply an extension and high-volume content population of human-made data models. This project situates data models as a key location in which the infrastructure of white-cisness, which I discuss later, is maintained. Broadly, I am investigating the ways that we think about data structures, use data structures in practice, and speculate about data structures.

My project contributes to emerging conversations that situate race and gender as central STS concerns. Scholars like Alondra Nelson (2016) and Dorothy Roberts (2009) have demonstrated the deep connection between race-informed scholarship and more "traditional" STS projects like the production of knowledge and the governance of emerging technologies. Their work goes beyond "adding" race or gender to analysis of sociotechnical concern; they combine foundational critical race theory with framings of research informed by issues of justice and social inequality. Building on their example, this project combines the insights from critical race and critical trans scholarship with the decades of critical work produced by STS touchstones like Judy Wajcman and James C. Scott. Furthermore, this work augments ideas of using algorithmic interventions to correct for disadvantage, offering instead opportunities for data modeling-focused intervention to correct for data models that might reinforce systemic benefit.

While this work's first audience is academic STS and critical data studies scholars, a secondary audience for a repackaged presentation is software engineers. Drawing on my experience as a software engineer and tech speaker, I envision presenting this work to engineers both in technical contexts and in more casual "engineering + society" contexts. With the structure of this project, I am pushing against the split between "engineering" and "everything else" (the humanities, social sciences). My methodological approaches trouble this separation and I offer a project in which engineering approaches can be social science approaches, and engineering methodologies—legible to industry and academic engineers—can be used to generate insights more commonly found within social science projects. Thinking about this division, this work can act as a bridge between social science/humanist and engineering audiences, engaging in work meaningful and familiar to both. Further, I believe that the identification of specific technical methods that mark and support white-cisness within data structures can open space for creative and engineering action.

The integration of detailed critiques of technology with conceptions of how racial systems work in the United States is underexplored in conventional STS scholarship and often within STS departments. Michael Mascarenhas, drawing on the work of sociologist Elijah Anderson, describes STS as a "white space": "white spaces are not just about *counting* but rather *accounting* for the ways that the dominant culture adopts a white habitus that is stratified by color" (Mascarenhas, 2018, p. 154). Mascarenhas performs some counting—indicting STS departments for their overwhelmingly white and (cis) male faculty.[4] However, he also performs some accounting—accounting for the ways in which we must acknowledge the "persistence of legacies of white male supremacy and Eurocentrism in the work of even progressive STS scholars" (Mascarenhas, 2018, p. 154). I follow this call for accounting in my work, performing a close examination of the legacies of supremacy in current data modeling and software engineering practices.

[4] Arizona State University's "STS Program" is my home department—the Human and Social Dimensions of Science and Technology (HSD) program. HSD fares well on (binary) gender lines, with 48% female faculty, but poorly along racial lines with 14% faculty of color, and 5% underrepresented faculty of color (which Mascarenhas categorizes as "Hispanic, Latino, African American, Native American and Pacific Islander"). However, HSD has a wide variety of affiliated faculty, and it is unclear how he is determining who "counts" as faculty within the department.

Critical Data and Algorithm Studies

Within critical studies of data and algorithms, I draw on several important conversations.[5] First, the field does important definitional work for terms like "data" and "database." As data becomes an increasing focus of academic study, the term can become dangerously universalized to the point of being less useful. I heed Angèle Christin's (2020) call to break the monolithic concept of data into its constituent parts in order to facilitate more precise study. In this work, I look not at individual datum, nor explicitly at any logics of "Big Data." To a limited extent, though I do engage with the power of data as representational form, I do so not from a semiotics perspective, but grounded within STS approaches. I follow Joanna Drucker's assertion that data is perhaps more accurately called *capta,* material that has been taken and constructed (Drucker, 2011). Approaching data as capta is in opposition to any work that assumes that data is an objective "given" to be gathered (Gitelman, 2013). When we reorient ourselves to viewing data as particular pieces of information sculpted and crafted to reinforce or support a view of the world, we are then better prepared to think of the models that demand that data as themselves constructed. In other words, the data models are the square holes training us to look for square pegs.

As Paul Dourish notes in his exploration of the materiality of databases, in many academic conversations about databases, the database is not clearly defined. In this work, I follow Dourish's useful three-part definition. A database is, first, a collection of data. This is often how the term is used by digital humanists and others thinking capaciously about things that can be "read" as databases (Thomer & Wickett, 2020). A database is, second, an organized collection of data items. Importantly, the data is organized according to a set and

[5] While I do not engage directly with foundational critical data studies scholarship in this project, my thinking here has been greatly informed by the works of boyd & Crawford (2012), Kitchen & Lauriault (2014), Kitchin (2017), Iliadis & Russo (2016), and Richterich (2018).

predefined schema, and all data in the collection conforms to the schema. Finally, a database is a database management system (DBMS). So, in Dourish's useful formulation: "a (3) database management system implements (2) a database, which organizes (1) a collection of data" (Dourish, 2014, p. 10). When I use database, I am collapsing all three aspects of this definition, and always assuming that a DBMS is involved.

My work diverges from much of software studies and the social study of technological harms in three important ways. First, I focus on not algorithms but on data structures. In addition to ontological explorations of the algorithm, scholarship shows us the social harms that algorithms can produce. In *Algorithms of Oppression*, Safiya Noble explored the ways that Google's search results—created by search algorithms powered by complex machine learning and artificial intelligence systems—reinforce systemic, and specifically anti-Black racism. Similarly, Virginia Eubanks (2018) demonstrated that the automation of social infrastructures like child protective services calcified existing inequalities into technological systems. These important works are tone-setting explorations into the human impacts of algorithmic choices. However, the noisy focus on algorithms often overshadows the importance of studying the data structures that support the algorithms themselves. I envision this project acting as a support to broader algorithmic studies works, offering tools for theorizing, reverse engineering and speculating about data models and I undergird my work with the important conclusion that **data models structure algorithmic opportunities**. Before algorithms can be written, tested, or deployed, there must be data models to support them and these models determine the shape, content, and fluidity of the data generated, gathered, and stored by software programs. Rather than engage in algorithmic audits and an exploration of algorithmic in/justice, here I am examining pre-algorithmic technological infrastructure.

Second, I focus not on the perpetuation of injustice, but on the maintenance of unearned benefit. Explorations of algorithmic (or more broadly, software) injustices frequently emphasize and detail the socio-economic-cultural injustices performed by sociotechnical systems. This is vital work, helping people who do not experience these harms to begin to understand how software can have widespread, lasting social impacts. However, the material conditions "brought to light" by this scholarship does not reveal anything that those experiencing it do not already know. From this privileged position, there is a risk of maintaining flattened assumptions about the lives of people systemically marginalized in the US today: those who do not in one or more ways fit within normative ideals of whiteness, heterosexuality, cisgenderness, non-disability, US citizenship, middle classness, etc. A focus solely on disadvantage disregards the structures and systems that create advantage (Hoffmann, 2019, p. 904). Structures of racial superiority are intertwined with structures of gender policing. As I have explained, the United States is a system run by white people for the benefit of white people. A benefit is defined as a profit, gain or advantage (though the source is not identified). An advantage, specifically, is defined as "a condition or circumstance that puts one in a favorable or superior position" ("Advantage, n.," n.d.). Thus, rather than point to non-whiteness and non-cisness as a problem, I point to whiteness and cisness as the issue itself.

Finally, by pointing to white-cisness as the site of unearned advantage, my work acknowledges the capaciousness of experience of those who do not experience that advantage. Importantly for me, that capaciousness includes experiences of joy and delight. In my work, and my life, I center the creation and pursuit of joy as a vital resistance to exhausting oppressive regimes.[6]

Rather than contribute to the reification of marginalized life as one of trauma and systemic violence (though there are many varied and valid cases why this link is also true) I want to move toward normalizing the view of white/cisness as one of unearned and continually reinforced benefit and ease. This shift opens space for the lives of those of us marginalized in one or more ways to be joyful and resistant and for our critical lens to shift towards white-cisness.

Infrastructure Studies

Foundationally, I follow Bowker, et al.'s description of infrastructure as "a broad category referring to *pervasive enabling resources*" (Bowker et al., n.d., p. 3 emphasis in original). Bowker et al.'s three-word description of the category of infrastructure is crucial to my work. The *pervasiveness* of infrastructure tells us that there is no point at which it ceases to exist or have influence. A pervasive infrastructure is one that is, in some senses, self-healing: fractures within it are mended by unfractured parts, maintaining its existence. *Enabling* has two important senses here. First, to enable means to cause to exist or to bring forth. Secondly, to enable means to make possible (i.e., to facilitate rather than originate). In both senses of the word, infrastructure *enables* the conditions of its politics. Perhaps most importantly, *resources* signals materiality and real-world support in a variety of forms. If infrastructures are value-laden and fundamentally relational (Star, 1999), what does this mean for our category of "pervasive enabling resources"?

[6] In this, I draw on the work of Brock (2020), brown (2017), Lu & Steele (2019), Muñoz (2009) and long legacies of Black, trans and Black transfemme resistance.

In this project, I treat infrastructure from both an infrastructural studies perspective and an engineering perspective. Both infrastructure studies and software engineering emphasize the products that emerge from *practice*. While driven by ideology, politics and values, my focus throughout is on the practices that contribute to the maintenance of infrastructures; to the practices that provide pervasive enabling resources. In computing, this includes not only the servers on which software are executed, but pervasive resources that enable continued development of computing knowledge. Computing infrastructure, then, also includes (non-exhaustively) university computer science departments, teach-yourself-to-code websites, organizations like Y-Combinator that incubate technology-focused startups, and open-source communities that maintain software. When we conceptualize computing infrastructure as encompassing pervasive enabling resources, we are then able to think more capaciously about the nature of that infrastructure.

The United States is built on the infrastructure of white-cis supremacy. To white-cis Americans, the white-cis supremacist infrastructure of the country is largely invisible and one of the fundamental characteristics of whiteness is its invisibility (Ahmed, 2007). Foregrounding the infrastructure of white-cis supremacy could be considered an "infrastructural inversion" (G. Bowker, 1994): the act of making visible that which is invisible. However, the infrastructure of white supremacy has always been visible to people who are not white. Philosopher Jonathan Flowers notes that the idea of "making visible" or "bringing to light" centers a white phenomenology.[7] It assumes a positionality of "discovering" or revealing that which is already experienced by millions and is epistemologically reminiscent of white settlers "discovering" North America. This is also

[7] Thanks to Liz Dietz for surfacing Flowers's work.

what Catherine D'Ignazio and Lauren Klein call the "privilege hazard": "the phenomenon that makes those who occupy the most privileged positions among us...so poorly equipped to recognize instances of oppression in the world" (D'Ignazio & Klein, 2020, p. 29).

In its relationality, the infrastructure of white supremacy creates a division between whiteness and the Other -- indeed, whiteness cannot exist without others it can point to as "not white". As David Roediger has argued, in the United States, Blackness has served as the primary boundary for whiteness (Roediger & others, 1999). The US constitution and its amendments—core infrastructural documents of the United States—concretize the positions of Black and Indigenous people. Notably, enslaved Black Americans were counted as three/fifths of a citizen for purposes of representation and taxation. Roxanne Dunbar-Oritz notes that Indigenous people were implicated in the second amendment to the constitution. The right to bear arms is rooted in the history of male settlers joining militias "for the purpose of raiding and razing Indigenous communities" and for joining "slave patrols" (Dunbar-Ortiz, 2014, p. 80). These patrols, a precursor to modern policing, were designed to surveil and police enslaved (Black) people who were attempting to move towards their own liberation. From its infancy, the infrastructure of white-cis supremacy was a system of pervasive resources enabling the control and monitoring of the movement of Black and Indigenous peoples.

Surveillance

"[w]hiteness wants to see everything but its own operations." (Fiske, 1998)

In the United States, surveillance has functioned as a system of racialized and gendered control (Browne, 2015; Lyon et al., 2007; Beauchamp, 2019). Historically, surveillance in the early United States was part and parcel of chattel slavery. From the design of ships to transport kidnapped Africans to early "slave patrols," surveillance in public has often been directed at non-white people (Benjamin, 2019a; Browne, 2015). Of course, public surveillance also reinforces gender normativity. In the US in the late nineteenth and early twentieth century, it was common for state and local legislation to enforce gender norms by requiring individuals to wear a certain number of items matching their "sex."[8] This focus on public surveillance does not mean that surveillance's racializing and gendering effects do not also happen at home. The Ring doorbell serves as a contemporary example of the ways that individuals invite surveillance into their houses.

As scholars identify sites of technological consequences, they locate these at what might be considered the periphery of society - determinations of criminality, border crossings, decisions about who gets WIC (Eubanks, 2018). These locations, rich sites of data making-gathering, may feel removed from the daily lives of many white Americans.[9] These sites also reinforce the relationships between marginalized people and society's edges. The contrast between the home and the jail cell is one that privileges perspectives uninformed by house arrests and case worker visits. It is a perspective that disregards the monitoring of WIC purchases and over-surveillance of "welfare mothers."[10] There is a surveillance and monitoring continuity between sites of administrative violence and the home, and in Chapter 3, I situate my investigation into the home as a nearly ubiquitous location.[11] I consider

[8] For a history of this law in San Francisco, see Sears (2015). For more on the surveillance of trans people, see Dinh (2003), Fischer (2019), Bender-Baird (2016).
[9] A series of surveys on the impacts of incarceration on families reveals that while only about 10% of white people have experienced over-one-year-long incarceration of family members, 31% of Black people and 29% of Indigenous people have Enns et al. (2019). Broadly 63% of Black people vs 42% of white people have experienced the incarceration of a relative.
[10] See Woodard & Mastin (2005) for an exploration of the racialization of "welfare mothers" in the context of other stereotypes about Black women.
[11] In addition to the works cited, my thinking about surveillance has been greatly influenced by Monahan (2010), Levy (2014), Genosko & Thompson (2016), Rapoport (2012), Green & Zurawski (2015).

"smart objects" like internet-connected sex toys, to be surveillance devices, invited into the home by the purchasers (Maalsen & Sadowski, 2019).

Whiteness and cisness in the United States

To be white in the United States is to be able to move through space without obstruction, and to move through the world without being forced to confront one's race (Ahmed, 2007). As Charles Mills notes, this ability implies the "existence of a system that not only privileges whites but is run by whites, *for white benefit*" (Mills, 2004, p. 31 emphasis added) What Mills is describing here he calls "white supremacy," though others have described it as "white privilege" (Sue, 2006) or, simply "whiteness" (Garner, 2007). The "white benefits" that Mills references can take many forms. Based on studies performed within the last 10 years, white benefits can be economic: the median white family has $143,000 more wealth than the median Black family (Institute for Policy Studies, n.d.); juridical: white arrestees are incarcerated at 20% the rate of Black arrestees (Rovner, n.d.). In one example, white children experience asthma death 500% less frequently than Black children (Lavizzo-Mourey & Williams, 2016). There is, of course, no biological reason that white children would experience less asthma. There are, however, *pervasive enabling resources* directed to keeping white children alive. Factors such as the locations of hospitals, the affordability of asthma medications, the quality of air, the economic and housing mobility are all part of the infrastructure of white-cis supremacy that benefits, in this example, white children. Of course, this is not necessarily an intentional choice by white parents, many of whom may still experience class or other marginalization. The system of white supremacy is not one related to the moral character or agency of the individuals experiencing the benefits or harms; it is a system of social control. Note that I intentionally choose and overload technology words—system, infrastructure, reification—to make concepts visible to engineering readers, and as an extension of Wendy H.K. Chun's work identifying race as a technology (Chun, 2013). Additionally, Tara McPherson (2013) has done a deft job of demonstrating the ways that spatial and social (read: racial) logics of gentrification influenced the architecture of modular operating systems like Unix. This is reminiscent of what Junaid Rana has termed "racial infrastructure": "a spatial formation in which the social, political, and economic relationships of racial systems operate through dominance and discursive power" (Rana, 2016, p. 113).

In her introduction and later clarifications of the term "intersectionality," Kimberlé Crenshaw points to the insufficiency of single-axis analyses (Crenshaw, 1989, p. 1, 1991). To look only at whiteness or only at cisness not only elides the ways that individuals can experience benefits from both systems, but also ignores their historical connection. A. Finn Enke emphasizes "how dramatically sex/gender congruence, legibility, and consistency within a binary gender system buy a privileged pass to social existence," especially when, among other things, one is white (Ænke, 2012, p. 64). Gender alignment (cisness) has historically been the restricted right of white individuals (Green & Bey, 2017), and as C. Riley Snorton argues, the creation of trans identity was ultimately a racial narrative requiring the erasure of African, Indigenous and pre-colonial gender identities (Snorton, 2017).

For the purposes of this project, I offer a parallel heuristic for gender, extending Mills' definition of supremacy to identify masculine supremacy and cis supremacy. Current social inequities, well-documented by feminist scholars, tell us of the existence of infrastructure that not only privileges masculinity but is run by men, *for male benefit*. This infrastructure of masculine supremacy (better known as "patriarchy") is responsible for, among myriad other things gender pay gaps and disparity in disease research. Useful here is that "masculine supremacy" does not force us to think in terms of binarist gender frameworks (of men vs women), but instead allows us to conceptualize a world in which benefits are received by "men" and not received by "everyone else." In other words, a world in which cisgender men control, and benefit from the control of, pervasive enabling resources directed towards their interests. Within a masculine supremacist framework, beneficiaries must be perceived to be conforming to masculine gender norms. To be viewed as outside those norms does not automatically mean that one experiences disadvantage, but rather that they are not the intended recipients of benefit.[12]

That there can be a loss of benefits for even perceived gender misalignment points to the existence of a system that not only privileges cisgenderness but is run by *cisgender* people, *for cisgender benefit.* This system is cis supremacy.[13] To be cisgender (or "cis") is to experience alignment between one's gender and the sex one was assigned at birth. Thus, cisgendered men who conform to gender norms are the express beneficiaries of masculine supremacy. There are, of course, men who are both cisgendered and gender non-conforming. This murky space—in which one can experience alignment but be perceived to be out of alignment—is why some trans scholars reject the use of the word cisgender as falsely universalizing. In his work on trans people and surveillance states, Toby Beauchamp declines to use "cisgender" because it "centralizes a form of gender privilege that emerges through normative race, class, sexuality and ability" (Beauchamp, 2019, p. 12). For Beauchamp, this centralization makes "cis" inadequate for discussing the repercussions of

[12] Within gender studies and queer studies, there is rich scholarship documenting the disparate treatment between masculine (often called butch) and feminine (femme) presenting women in queer communities. See Bauer (2008), Bergman (2009), Halberstam (1998), Levitt & Hiestand (2004), Rubin (2013).

[13] We can extend Mills definition to include disability, citizenship, reproductive status, and other labels of identity that mark one as non-normative.

gender deviations within state surveillance regimes because gender transgressive cisgendered individuals still receive harm. However, it is this centralization of gender privilege that is key here and that makes the term so useful; that cis people can be harmed by cis supremacy does not undermine its existence.

My opening example highlighted a white, European bias in the formation of name fields. Cis supremacy quickly becomes evident within data models and help us see the power of data models. Trans and gender-nonconforming people often experience friction when confronted with forms requiring gender selection. Recently, transgender activists asked to expand the number of gender options on websites, especially popular social media and online dating sites. In response, in 2014, Facebook increased the gender options that users can select from 2 (M and F) to 58, and this change was hailed as a win for all trans, gender transgressive and gender expansive users, finally able to use Facebook without having to lie about their gender identities.[14][15] However, what felt like a victory for trans activists was short lived. As a result, Facebook mapped all of these 58 genders back into a binary M/F in order to send data to their advertisers, who were not prepared to receive 58 different values for gender (Bivens, 2017).

The inability of marketers to receive specific gender information about Facebook users, frankly, seems like a feature and not a bug. I give this example not as a problematization of this data failure (as Rena Bivens notes in the work), but as an instance of the ways that data structures (and choices at the physical data level) can impact the ability of

[14] To be clear, some trans people feel included by binary M, F options because they do not wish to disclose their trans status and/or feel represented by simply an M or F.

[15] I use "trans" to describe the range of individuals who understand their gender to be out of alignment with the gender they were non-consensually assigned at birth. Gender transgressive and gender expansive individuals may share "trans" experiences or experience repercussions from their own (intentional or not) gender performances, but do not explicitly identify as trans.

software systems to perform inclusivity. Recently, popular dating application Tinder expanded their gender options, after transgender users reported experiencing high levels of harassment when using the app. However, as Anna Lauren Hoffmann notes, the expansion of data options does not immediately make a platform *safer* for its users. In Tinder's case, the expansion of gender options puts the responsibility on individuals to properly identify themselves: "such 'inclusive' solutions evade accountability and instead 'empower' individuals to regulate their own safety and visibility" (Hoffmann, 2020, p. 17). While Facebook's change may have been hailed as a win (and then a failure), and Tinder's as (in part) an abrogation of responsibility to its users—in both cases, the choices of data structures have material impacts on users.

Methods

In this dissertation, I use a variety of methods, several of which will be less familiar to the reader. As a result, I review two key methods at length—reverse engineering and critical speculative technical practice. In addition to these two engineering-focused methods, I also engage with narrative and autoethnography. All of my methodological practices are based in my ethos of community accountability.

Reverse engineering

Reverse engineering is, broadly, the process of uncovering how something was made.[16] Colloquially, to say "I reverse engineered it" implies that you were not the person who made "it" and that "it" is a thing whose contents, workings, dimensions, and materials

[16] Reverse engineering is not an idea new to computers. The first documented case of reverse engineering might have been Egyptians finding a chariot and trying to make one for themselves (Kahveci and Okutmuş 2015 in Neilson, 2020). During WWII, British and American troops reverse engineered German water cans, called Jerry cans (Srivastava, 2015).

can be known, or at least something akin to understood.[17] Within this project, I use reverse engineering as an umbrella term for activities designed to unpack or tease out the construction of a technological system (Dang, 2014). These activities are performed to help those who did not create the software discover "the true purpose and actions of code" (Dolan-Gavitt et al., 2015).

In an early IEEE piece on the philosophy and method of reverse engineering, M. G. Rekoff (1985) compares the process of reverse engineering to criminal detective work, gathering clues to infer intent and reconstruct motivations and actions (but not necessarily recreate the product). When working outside of a reproductive goal, there are no set steps involved in reverse engineering, nor a universal point at which the process is complete. In his discussion of reverse engineering, Robert W. Gehl defines it as "a method of producing knowledge by dissociating human-made artifacts" (Gehl, 2015). It is an iterative unfolding of the layers involved in the software and the specific orientations of the engineer. For example, a security-oriented reverse engineering project might take a security-penetration approach: looking at all available points of vulnerability and developing tools to insert malicious code into the system. A performance-oriented reverse engineering project might take a stress-test approach: overloading various subsystems, watching errors, and decoupling systems when they fail.[18] For this project, reverse engineering is the disassembly of software with the intention of learning more about one or more aspects of the system and without a reproductive end goal.

[17] Whether or not a reverse engineered object can be known and understood is an ontological and epistemological question that I leave to others.
[18] Indeed, there is robust scholarship on different approaches to reverse engineering. For reverse engineering design patterns see Shi & Olsson (2006), Dwivedi, et al. (2018); for deobfuscation of code, see Udupa, et al. (2005), Yadegari, et al. (2015). Many innovations within reverse engineering processes were developed in the IEEE Working Conference on Reverse Engineering (1993 – 2013).

This approach is in conversation with work around what Ruha Benjamin calls "critical race code studies" a combination of critical race studies and STS insights to look at what she describes as "The New Jim Code"—a sociotechnical regime of injustice towards Black Americans. She relies on the anthropological practice of "thin description" as a way to embrace the uncertainty and unknowability of systems that are, in part, within anti-Black black boxes (Benjamin, 2019b). The literature of reverse engineering may not use the same terms, but it is also one of unknowing. Engineers use reverse engineering techniques specifically because they cannot know how a system was constructed. Reverse engineering is a "technical-first" set of practices. This makes it highly complementary to the work done by STS, critical algorithm, and critical software researchers. Although these works often start with the social impacts of a technology and work backwards to find a source, reverse engineering (despite its name) starts with the artifact working forwards to understand how a system produced its results.

My form of reverse engineering (as described in Chapter 3) is particularly goal-oriented, as I am working to discover the physical data model within the software.[19] However, it is also expansive. Rather than beginning with the technical specifications, I begin the reverse engineering with close reading. Understanding the product and the way it describes itself is crucial to my reverse engineering methodology.

Critical Speculative Technical Practice

Within the discipline of design, there is a long legacy of critical design practice. Coming out of the Italian design movement, designers put ironic or satirical spins on

[19] Premerlani & Blaha (1993) introduced a reverse engineering methodology specifically for relational databases. Theirs is an important early work, though not accessible for this project because I did not have access to the software's database.

common objects to defamiliarize them and provoke deeper viewer engagement with the socio-techno-political factors of the moment in which the design exists. (Malpass, 2013, 2017).

Terms like speculative design, critical design, experimental design, design fictions all point to the same general practice: designing for worlds that do not yet exist (but perhaps should). According to James Auguer, regardless of the term used, there are three important characteristics to this speculative practice: “remove the constraints from the commercial sector that define normative design processes; use models and prototypes at the heart of the enquiry; and use fiction to present alternative products, systems or worlds” (Auger, 2013, p. 1). Design objects, those artifacts produced by speculative design, can themselves “co-produce knowledge, by working through complex design problem spaces in non-reductive ways, proposing new connections and distinctions, and embodying design ideas and processes across time and minds” (Bardzell et al., 2015, p. 1). The boundaries of design practices have been variously defined to include technological practices like data modeling, programming, or interface design. For my purposes, creating a false division between design practices and software engineering practices is unproductive and ahistorical. In Chapter 4, I perform critical speculative work using the tools of (software) engineering in addition to tools of design.

Theorists of speculative methods insist that the objects of speculation must be removed from capitalist gains; however, there is no clear method for extracting one's work or one's mental model of work from capitalist paradigms. According to Sasha Costanza-Chock, we are all designers; everyone designs. However, the power dynamics arise when individuals get paid to design - when design as a task undergoes professionalization

(Costanza-Chock, 2018).[20] Corporations and capitalism have had an immeasurable impact on the development of both engineering, (software engineering) and design.

Rather than ground my work squarely in the literature of design, I take Phil Agre's critical technical practice and combine it with speculative design ideas to form a critical speculative technical practice. In this work, I consider speculation to be a methodological approach - speculation as method has a set of epistemological foundations that influence my choices. Rather than work in a future in which everything is flexible, or a world influenced by science fiction, I am not working to speculate at the far future, but to speculate at technology for the very near future—technologies that could be implemented now, given our current infrastructure. In this way, I am not working outside of capitalist expectations, but *within them.* Rather than build for futures that may come, I am thinking about futures that are likely to appear, and the ways that I can transform them slightly but significantly—to bias them towards my own values, objectives and priorities.

Autoethnography and community accountability

My work in this dissertation is in conversation with the communities to whom I am accountable. Having a critical technical practice and a strong near-term temporal orientation means that my work is in conversation with people living and making in the present with the tools we all have. I have long experience working within design and technical spaces as a software engineer for 20 years.

While I write from the position of a tech insider, I also bring a perspective informed by my social location. My interest in white supremacy and cisnormativity is informed by my

[20] This separation into the professionalization of the work is reminiscent of David Noble's work on the professionalization of engineers in the United States (Noble, 1979). His work highlights the role of corporations in the creation of the white-collar engineer

identities as a white, queer, trans person in the United States. My mother is a first-generation American, with both of her parents coming from Italy. She raised me as single parent in Las Vegas, Nevada where she worked as a casino dealer. I never met my father, who is now deceased. His family has been in the United States for centuries and is filled with generations of doctors and high-ranking military men in the US South. I assume the worst about my father's family (an extended family that declined further contact with me when they found out that I am neither cisgender nor heterosexual) and assume that that their generational wealth was accumulated not only by way of being white in the US, but also by participating in the enslavement of kidnapped Africans and their descendants. My experience of whiteness colors my experience of every other aspect of my identity, and I treat it as both a banal factor of my existence, and a powerful location of advantage that, for many years, I was unaware that I was carrying.

I am a queer trans person. I do not personally experience alignment with masculinity, femininity or androgyny, and decline to call myself a trans masculine or a butch person. I am, ultimately, a person who wishes to have no gender at all -- the idea that a person can feel steady and aligned in their gender is a wildly foreign concept to me. Ultimately, any medico-legal-social challenges I experience via queer-transness are lessened and transformed by my whiteness, US citizenship, "unaccented" English, and lack of visible disabilities or illnesses. Given the tendency for white trans people to center their oppression in their personal narratives, I believe it is important to note that my transness is racialized and my gender illegibility creates more conditions for livability because of my whiteness (and other normative aspects of my social location). I am undertaking this work from a point of tension, as I experience both benefit and harm from cis-white supremacist infrastructures.

I will likely not see the destruction of white-cis supremacy in my lifetime. I am grateful to the scholars dreaming abolitionist futures, and also acknowledge that those futures are not implementable (by me) at this moment. The communities to whom I am accountable deserve both inspiring visions of futures and near-presents we could be having, as well as recommendations for real, implementable solutions.

There is, of course, a tension between approaches that are radical and innovative and those that are implementable given current technical and social limitations. Because I feel accountable to communities of makers, as well as communities of academics, I find myself continually navigating the tension between radical and visionary theoretical approaches and those that could be built tomorrow by someone with an appropriate technical skillset. As a person who came out of software engineering communities and open-source communities and leaving those communities to get an education and produce knowledge that is not *usable* to the people within those communities is in violation of my values.

Outline of Chapters

Working through an extended code example, in Chapter 1, I demonstrate the interrelationship between data models, algorithmic approaches, and written code. I argue that data models structure algorithmic opportunities. Without certain configurations of data, algorithms cannot function, and I identify data models and data structures as software components requiring deeper scholarly study.

In Chapter 2, I review pre-relational database structures, like hierarchical and network databases. Because of the rigid structured nature of these databases, all databases of a certain type were similar and no particularly advanced levels of visualization were required.

As a result, the introduction of the relational paradigm was a key moment. As the relational model grew in popularity, so did the techniques of creating and visualizing data models.

Furthermore, I identify techniques involved in data modeling that uphold white-cisness. These moves happen in data models, and the chapter concludes with an examination of the relationship between data models and built software.

In Chapter 3, I situate my inquiry in the data-driven domain of surveillance devices. In the United States, surveillance is a key mechanism for the maintenance of whiteness and cisness. Government and police surveillance co-produces Blackness while medical and social surveillance co-produce transness; together infrastructures of white-cisness are upheld. The continuum of surveillance locations extends into the home, and I argue, the surveilled home is a key site for the maintenance of white-cisness.

I reverse engineer the data model of a series of home sexual surveillance devices. Reverse engineering is an umbrella term for a constellation of activities designed to unpack or tease out the construction of a technological system. I outline four key steps to reverse engineering Android applications and demonstrate the artifacts that result from each step. After penetrating the application and examining the code, I demonstrate the inferential creation of physical and conceptual data models.

Rather than tidily point to aspects of the data model and speculate about the generalization or abstraction therein, I offer a vignette to illustrate the messy entanglements that occur within datafied white-cis surveillant infrastructures. Within the application, I demonstrate the ways that reification, generalization and decontextualization work together within these devices to promote and maintain world views consonant with white-cisness.

In Chapter 4, I engage in grounded speculative engineering work, informed by my critical technical practice. I discuss the ontological commitments of relational data modeling, specifically the choice of what concepts and objects are promoted to entities within the model. Using trans lives as a starting point, I explore the speculative possibilities of using reification to create more livable and workable data models for gender. I consider the disconnect between the way that gender is represented in data models and lived experiences of gender. Speculative engineering is the work of designing for worlds that do not yet exist (but perhaps should). Informed by the technical moves in Chapter 2 and the result of reverse engineering, I offer a data ecosystem that neither elides nor cements gender.

CHAPTER 1

MAKING SOFTWARE

In the mid-1960s, before E.F. Codd introduced the relational paradigm that would shift databases and data models, programming was experiencing a "crisis." Large software projects, usually sponsored by the military, were chaotic, behind schedule and over budget. Programming projects needed more structure, and that structure started in part with a rebranding of the task itself.

In 1968, the NATO Science Committee sponsored a conference that created the term "software engineering," to elevate the creation of software to the status of other white collar (mechanical, electrical) engineering tasks. One attendee noted that "Programming is still too much of an artistic endeavour" and advocated for greater structure in the education and practice of the work. Another decried the language used to talk about software --- words like "elegant" and "powerful" were too "fuzzy" for their vision of what software engineering could become (Gehl & Bell, 2012, p. 11).[21] Like other engineering tasks with discrete steps, as a result of this conference and other meetings like it, software engineering also became a discipline with discrete steps designed to control the project and make it appear predictable and controllable.

In this chapter, I describe the discrete and "repeatable" steps that comprise one common approach to software engineering - the software development lifecycle (SDLC). Understanding these steps is an important first step to recognizing the importance of data modeling in the outcomes of software. The SDLC illustrates the connections between data

[21] There is still debate over whether software engineering is a proper subset of engineering. See Davis (1991) for one perspective.

models, databases, programming choices, and the resultant software.[22] In order to understand the importance of data models, we must understand the lifecycle that is used to make software, and where data models sit within that lifecycle. The ways of making software are invisible to most, and data models in particular can be a difficult thing to conceptualize as a discrete phase of software development.

After discussing the SDLC, I move through a technical example of the ways that data models and data structures impact code. Through small code examples I illustrate the ways that data structures create algorithmic opportunity, a crucial part of my argument that data models require far more scholarly inquiry than they are receiving at present.

The Software Development Lifecycle

Software is made, maintained, and end-of-lifed through a near endless process---sometimes by the team(s) that created it, sometimes through abandonment or server shutdowns and data destruction. Thinking about software as part of a process, the SDLC, is important for several reasons. First, the SDLC is a reminder that software is created by people in a highly iterative process into which intervention is possible at any moment.[23] Second, it directs our attention to the ways that software "products" unfold and are created over time, an important lever for analysis here. The creation of software depends on linear and teleological notions of time - perhaps without those, software creation (and even software idealization) could not exist). Third, the SDLC offers an opportunity for the uninitiated reader to learn the steps involved in the creation of a "software product" and the ways that each step is itself a process. Finally, the SDLC aligns with descriptions of violence

[22] In this work I use the term "software engineering" to refer to all tasks required to produce software, and programming to refer specifically to the act of typing code.
[23] I explore one form of possible intervention in Chapter 4.

as processes --- not as one-time events, but as experiences and structures continually unfolding to do harm (Hoffmann, 2020).

The steps of the SDLC are variously conceptualized, but generally break down into three categories: planning and design; building and testing; and publishing and maintaining.[24] Importantly, although these steps are linear (one cannot publish before programming, nor program before at least a little bit of design) they are not tied to a particular time domain or iteration cycle.[25] This means that a team could complete a full SDLC in a weekend or take many months for only one step. For my purposes, I use the version SDLC that I used during my software engineering career. As I step through the process below, I draw ethnographically on that experience and describe steps as I participated in them. This narration happens "from the inside," informed by the influence of Phil Agre's critical technical practice.

The six steps of the SDLC are Plan, Analyze, Design, Implement, Test & Integrate, Maintain (refer to Figure 1).[26] For the simplicity of the text below, I will write as though the software is being engineered by an internal teams made of full-time employees and that each team has a sub-team to lead it. Thus the "Software Team" comprises a Planning Team, a Design Team, an Implementation Team, etc. It is worth noting that on many projects, there are no individuals who are on more than one team other than high-level project managers

[24] See Ruparelia (2010) for a review of other ways of modeling the software development lifecycle.

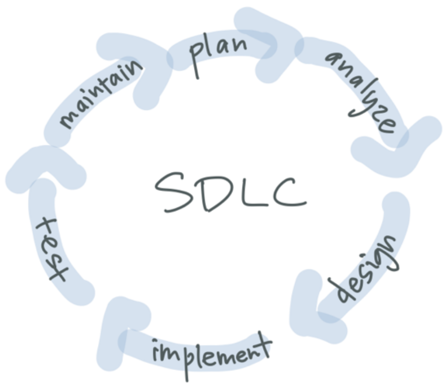
[25] This signals an important separation between the development *lifecycle* and the development *methodology* The lifecycle indicates the steps that one must complete, and the methodology determines how those steps are arranged with respect to time and staff. Readers might be familiar with terms like "agile," "standup," "extreme programming," or "waterfall."

[26] Throughout the project, I use hand-drawn diagrams rather than computer-generated diagrams. This is a deliberate choice to emphasize the human and individual within the software creation process. Indeed, "the visual representation [of an engineering concept] is a re-presentation, and not the thing itself" (Frankel & DePace, 2012, p. 3). Thus my manual diagrams and illustrations are not meant to belie the computational nature of the content, but to amplify the human involvement. My thinking here is also influenced by creative computation scholars like Taeyoon Choi and others making computational works out of fugitive materials.

overseeing the entire project. When work passes from one phase to another, it changes hands entirely. This is an important consideration when thinking about who can meaningfully intervene in an engineering problem with ethical stakes (which is nearly all engineering problems). The division of labor implied by the cycle is by design rather than by necessity. Teams of one create working and successful software products and/or packages (the open-source ecosystem is an excellent example of this) and in that situation one person works as designer, implementer, tester. The larger the product, and the more person-hours required the larger the team will need to be. As early military-industrial project leaders discovered, these larger teams were difficult to manage in several ways—including communicating the vision of the product itself. The Garmish conference made an explicit move to separate software engineering from artistic endeavors and to emphasize that software engineers needed less creativity, not more.

Figure 1

The software development lifecycle.



Plan

The planning phase is the initial phase of a project, designed to facilitate gathering the project requirements and constraints. In the planning phase, the Planning Team may meet with both internal and external stakeholders A stakeholder is anyone with a "stake" in the final product. Identifying those who have a "stake" is an arbitrary decision, ultimately determined by those in power in the process. Stakeholders are frequently those with a financial interest in the product, though stakes could be emotional, geographical, and/or political. Within technology production, there is a recent move towards thinking broadly about who a stakeholder might be and including them as early and as frequently in the process as possible. As Rozilawati Razali and Fares Anwar note, the identification of and engagement with stakeholders is often unstructured and unclear. Thus, teams should choose a specific methodology for identifying and engaging with stakeholders (Razali & Anwar, 2011). Deciding who is a stakeholder and which stakeholder voices will be heard is a highly political process that can have a significant impact on the final product.

The planning phase is often described as "non-technical" and while budgets and timelines may be involved at this point, they are not examined in detail. During planning, the Planning Team may work with other departments to make sure there are available resources (this is usually another name for people), and to ensure that the project has a mostly appropriate budget with a mostly doable timeline. A successful planning phase will deliver everything needed to the Analysis Team.

Analyze

During this phase, the Analysis Team takes everything delivered from the Planning Team and begins to make sense of it. Invariably, there will be conflicting requirements. For example, it is common to have a design stakeholder who insists that the product use flashy colors and innovative display mechanisms and an accessibility stakeholder who needs simple high-contrast colors. The Analysis Team turns stakeholder interviews (aka, qualitative data) into structured requirements. The format of structured requirements varies from team to team, but it is common to use a format called "user stories." User stories is a method of gathering requirements and formatting them so that everyone reading understands the context behind the requirement. User stories take the following format: "As a [type of user] I should be able to [task] so that [business reason]". User stories might include statements as general as "As a mobile user, the page should load in under 5 seconds so that I don't have to wait." or as specific as "As a logged in user, I should be required to update my password every 60 days but not more frequently than every 24 hours in order to meet corporate security rules." For a sense of scale, it's not uncommon for projects to have hundreds of user stories if that's the format they are using, or to have thousands of requirements.

In the same way that the choice of stakeholders can influence the contents of the user stories, so too will the user stories influence the designs created in the next phase. However, the Analysis phase does not simply create requirements, it also prioritizes them. This process is frequently a part of requirements engineering (RE). RE is the highly political process of deciding which stakeholders' requirements will be built during the current production iteration. The prioritization is an imperfect process that is difficult to standardize or to recommend a universally good approach (Hujainah et al., 2018).

Design

When people hear the word design, they may think of activities that involve images, shapes, or colors - designing layouts, graphics, brand images, color schemes. While the visual design of a software product may be designed during this phase, the data design / database design also occurs here. If this phase includes visual design (which it often does), a Lead Designer and their design subordinates will work to put together a series of deliverables for stakeholder approval. The Design Team might provide artifacts like mood boards, wireframes, and comps. They will then iterate on these artifacts until they have stakeholder approval. This set of stakeholders might be internal, might not. Frequently, visual design choices may impact functionality, so it is common for the Implementation Team to work closely with the Visual Design Team to discuss what is possible.

Design also includes aspects of the software that are not visual, like infrastructure and data. Depending on the product's scale, there might be an Infrastructure Design Team. That team will design the arrangement of servers, load balancers, firewalls, backups/redundancy, and other storage and security measures required by the Analysis Team. There will also be a Data Design Team. This team will create data models, separate from the physical storage requirements of the database(s) in use. *The data models created by a Data Design team is the focus of this dissertation.* Like the Visual Design Team, the Data Design Team's work is subject to revisions by stakeholders and further refinement through iteration and conversation. Thus, like all software artifacts, it represents not only the clear vision of the designers, but a series of compromises with others.

Implement

During the Implementation phase, the implementers (usually called programmers, developers, or software engineers), build it. They spend their days writing code and meeting small milestones and accomplishing small tasks, as determined by the Lead Developer, Technical Architect and/or Project Manager. (The exact arrangement of Person-making-tasks varies in each organization). In implementation, the visual designs become code to render them, the infrastructure design becomes provisioned servers, and the data design becomes databases. The implementation phase is usually what most resembles "writing code" in the popular imaginary. If the project was using user stories, the Implementation Team will frequently complete one user story (or set of requirements) at a time, building functionality with the context provided by the Analysis Team. In the next section, I explore the impact of the Data Design team's work on the implementation of a software product.

Test

During the testing and integration phase, the work done by the Implementation Team is tested for conformity to the requirements. In most cases, testing is called Quality Assurance (QA). There are several common kinds of testing: white box testing, in which the QA team has access to the code; black box testing, in which the team tests the software without access to the code or server, treating it like a "black box"; smoke testing, in which it is assumed "If there's smoke, there's fire" and the QA team will test basic functionality or core user pathways like user registration, or shop checkout. Members from the QA team may have been involved with the project from nearly the beginning, and they often help to structure user stories and requirements in ways that make testing easier.

It is important to remember that QA teams are not testing for "quality" but for a software's ability to conform to the requirements as written. This is a crucial distinction. If a QA team member finds that when software conforms to the requirements it has unintended consequences, that team member may be able to report it to the Implementation Team, but also will likely not have any power to ensure that a change is made. In other words, by the time software reaches QA, it is often too late to screen for ethical or social consequences of a software product. Those consequences are best caught much earlier in the SDLC.

Maintain

It is during the maintenance phase that fixes are released (or "bugs" "patched"), and new features developed and deployed. In many cases, once a product is released and stable, the maintenance overhead reduces enough to require very little time and only automated monitoring. During the maintenance phase, significant new features will be developed using the SDLC and then merged into the existing software product. For example, when a platform like Twitter releases a new feature like the downvote button (new as of this writing in 2022), that feature will have gone through every phase in the SDLC before it was released into production.

While the phases may seem self-evident, or otherwise uninteresting, it is crucial to put data models in their rightful place within the SDLC – that is, they are created early in the process and have immense power to impact everything that comes after. Depending on the structure of the software team, there might be no overlap between people working on the Planning, Analysis, Design or Implementation. There is no structured way for individuals to pass their ethical or ideological orientations down through the process, except through the artifacts they hand off. Considering the SDLC, it becomes clear that there is not *one* person to blame when software has negative social consequences.

Data Models Structure Databases

Entity relationship diagrams (which I discuss further in Chapter 2) allow developers to work across all three levels of data models---conceptual, logical, and physical. At the highest or most abstract level, called the "conceptual level," the conceptual model contains entities.[27] Sometimes, modelers are modeling the world as they know it to be (e.g., <u>customers</u>

[27] This is an oversimplification. For a good introductory text on how data modeling works, see Simsion & Witt (2005).

place <u>orders</u> for <u>products</u> in Figure 2) and other times they may be modeling a speculative world that does not yet exist (<u>astronauts</u> use their own <u>spaceships</u> to lead <u>space tours</u>). A logical model focuses on the ways that entities are related to each other; a conceptual model becomes a logical model when detail is added and when relationships between entities are made explicit. Beginning with Figure 2, a conceptual model containing customers, orders and products becomes a logical model when we add further detail about each entity (like attributes) and how they are related. (See Figure 3). A logical model becomes a physical model when field-level information is added. That is, when information about the attributes of an entity become specific data types and fields. The physical model, often regarded as the lowest level, contains all information necessary to create a database to contain and enact the model (see Figure 4). At each level of data modeling, database designers are making choices about what entities and relationships are most important and worth storing. No data exists yet, but the containers in which data will be stored are being formed. It is important to remember these three levels are all included when the term "data model" is used (and will be included here as well). In Chapter 3, when I reverse engineer application data models, I am working across the levels, and most frequently diagramming the physical level.

Figure 2

A conceptual model for a very small order system.

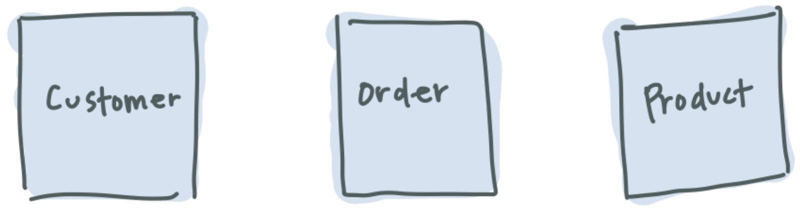


Figure 3

A logical model for a very small order system.

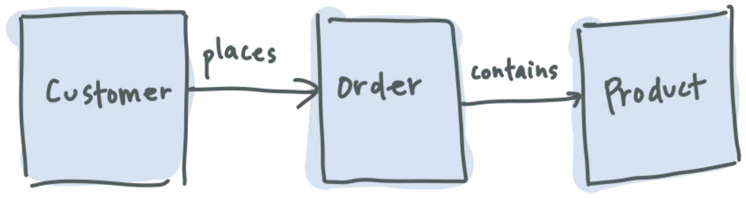
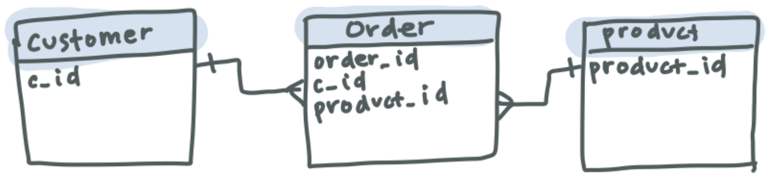


Figure 4

A physical model for a very small order system.



At each level, database designers are refining the entities, relationships and fields contained in the data model. Choices made at the conceptual level (to make order its own entity, for example) cascade down to the logical and physical levels. The physical model and the resulting database are a direct result of the choices made at the conceptual level. This means that when we consider something like dataset bias, our consideration is incomplete unless we examine the data models shaping the dataset. Thinking about dataset bias through the lens of data model levels expands the field of analysis and provides us with more opportunities to think about how data and datasets are shaped. The majority of contents of a data model (its entity, field, and relationship) are determined by the social, cultural and commercial conditions of its use. However, there are some fields that have become standard or expected because they provide expected computational tools when included. For example, it is now commonplace for entities to have fields like created_date because this field allows for sorting records based on which might be oldest or newest. It also allows developers to write code to purge old records, and for computer systems to effectively synchronize distributed storage. This is an obvious yet important point to underscore—field creation and its attendant constraints are entirely created by the data modeling and development teams. There is no general computational or operational advantage to having, for example, first name and last name fields instead of given name, middle name, first surname, and second surname fields. In other words, the problems created by a database's gender, name, and other identity fields are not simply software problems, but reflective of assumptions embedded within the broader sociotechnical systems. It is an easy STS truism that software systems are also sociotechnical system and that a system like, for example, Facebook encompasses not only the code and data used to power it but also the cultural norms accompanying the creation of the code (Grodzinsky et al., 2012). Within any software system, designers (programmers, architects) made assumptions about the society and the individuals who were their concern. In the United States, the included cultural norms and social assumptions made generally include the fundamental centering of normative identities---including whiteness, cisgenderness, maleness.

Databases Structure Software

In the same way that data does not exist without context, data modeling also happens within the governmental, industrial and nonprofit contexts of software production. Data models, of course, are not created for their own sake, but as part of software engineering, that process that is "intended to support professional software development rather than individual programming." The data models I consider and the processes I examine are all squarely within this realm of "professional software development." In this work, software is not simply another word for a computer program (or set of instructions) but "also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful" (Sommerville, 2015, p. 19).[28]

Software is, and has always been, about more than algorithms and their code. In a germinal computer science text, Niklaus Wirth asserts that software programs are actually algorithms + data structures (Wirth, 1976). In an extension of Wirth's work, Paul Dourish supports Wirth's elevation of data structure to an algorithmic peer in terms of the necessity of analysis, noting that we must analyze algorithms "alongside data, data structure, program, process, and other analytic entities" (Dourish, 2016, p. 2). Notice that Wirth's focus is on the data structures rather than the data itself---algorithms are designed to work with "forms and regularities" rather than specific pieces of content. Algorithms are productive, taking inputs (data) and generating conclusions, insights, verdicts, and diagnoses. However, the contours and capabilities of those inputs are directly determined by the design of the data structures.

[28] For my purposes, "software" *does* include all of the artifacts and configuration required to create, use and maintain a software program. When I am referring to an executable binary, I will use the word "program."

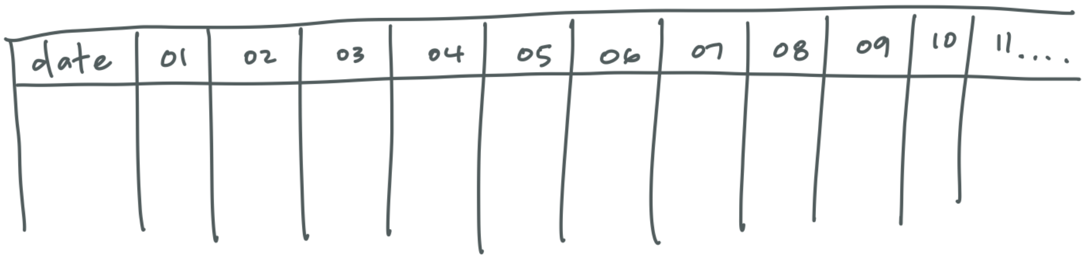
While it is common to hear phrases like "garbage in, garbage out" when talking about algorithms and datasets, the data models prefigure the "garbage" data people refer to—they determine the shape of the containers before they are filled with any data at all.

Data models become data structures, and data structures control what algorithms can be used. The relationship between data structures and data models can be conceptualized as a linear one: data structures are data models translated into programming code. Data models are visualizations and abstractions of the ways that data gets stored and interrelated. Those data models, when moved into code, become data structures. Consider the following small programming task: I need to build a simple weather application that displays the hourly weather and reports the high and low temperature for the day. As the programmer and data modeler, I need to decide how to store the weather for each of the 24 hours in a day. One approach to designing the data model might look like Figure 2.[29]

[29] Technically, this is a table and not a data model. Further explication of data models will come in Chapter 2.

Figure 5

A sample data table for a small weather application



A data design like the one in Figure 2 has a direct impact on the code that I would write. For example, based on this table, I might create variables for each day's weather. Note that lines starting with // are comments to guide the reader and not executable code.[30]

```
// Store the weather for each hour
var midnight = 44;
var oneam = 42;
var twoam = 37;
var threeam = 36;
var fouram = 40;
var fiveam = 36;
var sixam = 37;
 ...
var elevenpm = 43;
```

The arrangement above allows me to store 24 hours of temperatures, though it is inefficient in many ways. Additionally, I need to be able to determine the day's highest and lowest temperatures. To do this, I could compare each hour to each other hour one at a time:

```
// Find the coldest hour of the day.
var coldesttemp = midnight;
if (coldesttemp > oneam) {
   coldesttemp = oneam;
}
if (coldestttemp > twoam) {
   coldesttemp = twoam;
}
```

[30] I am using "vanilla" JavaScript for all of the code samples in this chapter because I think it is the easiest to read for those without programming experience. Note that JavaScript is a dynamically typed language, which means that data types for variables to not need to be declared in advance.

However, this is wildly inefficient, will not support more than a few variables, and is an error-prone approach. This example illustrates that my choice of data storage (a collection of unrelated variables) has a direct impact on the operations we can perform with our data (i.e., on the algorithms available).

I might instead opt to store the day's worth of weather data in a data structure called an array, which is a container "that hold[s] a fixed number of items of a particular type." (Smith, n.d.). Arrays are data structures that allow data items to be stored together and selected by a numeric index.[31]

```
// store weather in an array
var temps = [44, 42, 37, 36, 40, 36, 37];
```

The programmer is not required to use an array, but arrays make several programming tasks easier. With an array, a programmer can use an algorithm to iterate over items to perform the same operation on each one. In other words, the choice to store data in an array, versus not in an array, has a direct impact on what a block of code (aka, an algorithm) can do. If I choose to use an array, I can use a bubble sort (or other sorting algorithm) to determine the day's highest and lowest temperatures. For example:

```
for(var x = 0; x < temps.length; x++){

    // Loop through elements we haven't yet seen
    for(var y = 0; y < (temps.length - x - 1); y++){

        // Is the item we're on bigger than the next item?
        if(temps[y] > temps[y+1]){

        // If this code is executing, then it is!
        // store the big value in a temporary variable
        var tempVariable = temps[y]

        // move the littler value into the slot we're on
        temps[y] = temps[y + 1]

        // move the bigger value into the next slot.
```

[31] Arrays that allow for items to be referenced by a string are multidimensional arrays, and not available in all languages.

```
            temps[y+1] = tempVariable
            }
        }
    }
```

As we know, the choice of algorithm can have a powerful impact on the outcome However, as this very small example demonstrates, those algorithms cannot operate without the proper data structures in place. In these examples, the data structures are simple containers (i.e., arrays) to hold data of type number. However, in large systems, there is an immense volume of data items that will need to be structured and stored and it is insufficient to think at the data structure level. Thus, we must consider the *data models* that shape the data structures.

In this chapter, I reviewed the process of creating software, the SDLC, and illustrated one way in which data models and data structures create algorithmic possibility. The steps of the SDLC and the code examples are foundational to my argument that data models are important site of analysis for those of us concerned about the social and cultural impacts of software systems. In the next Chapter, I offer a critical analysis of some of the techniques of data modeling, illustrating the ways that data models are both important sites of analysis and mechanisms that mirror societal structures of power.

CHAPTER 2

THE EPISTEMOLOGY OF MODELING

In the United States, the 1970s were an active period for political and social movements. The rise of counterculture movements and emergence of revolutionary politics did not skip the computing culture of the time. The 1970s also saw a "database revolution" (Kerssens, 2018). One important ingredient in this revolution was the introduction of the relational paradigm of database development, modeling, and querying. Prior to the database revolution, data was treated as material to be processed rather than collected, and computers as the tools doing the processing. The affordances of available technology shaped this assembly-line like perception of data and data processing. Software was written for specific sets of computer hardware and updating that software was tedious and time-consuming. The data storage paradigms popular in the late 1960s and early 1970s—hierarchical and network databases—were not conducive to storing large volumes of data for future unknown processing tasks.[32]

This database revolution ushered in a "complete psychological reorientation" to the uses of computers and their data storage. At the time, proponents of this revolution advocated for a "database approach:" "a view of the database as not just a technology, but as a completely new managerial philosophy and mindset." (Kerssens, 2018, p. 14) This new mindset was facilitated by the introduction of database management systems (DBMS) and the relational database (Haigh, 2009). DBMSes provided ways to interact with databases that made databases, data processing and data storage more accessible to computer users. DBMSes thus helped to reinforce the computer's utility as an administrative tool. Indeed,

[32] In other words, there was little room for an approach like "Let's collect it in case we need it, and figure out what to do with it later." This is an approach that we might now describe as a "big data" strategy

databases began to have significant impact on the ways software was developed and "the materialities of database technologies shape[d] the kinds of 'databasing' that [could] be done and imagined" (Dourish, 2014, p. 39). As "databasing" evolved, so too did the paradigms used to conceptualize and visualize the databases. The relational database and its convenience and flexibility required programmers to use new tools to communicate database design and architecture. In many ways, data modeling—that crucial act of naming the things to which a software system must attend and a database must store—evolved out of new "databasing" and the shift from data processing to data collection.

In this chapter, I briefly review pre-relational database paradigms, contrasting them to the innovation of the relational database. The relational model, and the subsequent data modeling developments, are based on a fundamental epistemological claim—that data has a natural structure in which it is best to be both stored and visualized. I interrogate this claim and link it to data modeling techniques. I demonstrate that each technique is a way that data models mirror social power structures, specifically white-cis supremacy.

Before data modeling

Data models are, loosely, a tool to help software engineers track and visualize the things (data) that an application needs to attend to.[33] The modeling process requires the modelers to draw boundaries around the information that is relevant, and to decide what to include in a data model. As I discussed in Chapter 1, data models are often made before (or

[33] Data modeling is considered a type of conceptual modeling, in which parts of the "universe of discourse" are abstracted to be stored. Conceptual modeling as a field was named by John Mylopoulos in 1980. He describes conceptual modeling as the activity of constructing abstract models of knowledge about some world uses it synonymously with the terms "knowledge representation" and "semantic data model" as they have been used in artificial intelligence and database discussions respectively (Mylopoulos, 1980, p. 167).

in tight parallel to algorithmic and code decisions and play a significant role in shaping how a given piece of software works.

While it is not the case that data modeling never happened before relational databases, it was far less common. Data modeling as a software practice was ushered in by the relational database paradigm. Prior to relational databases becoming popular, the structure of the data was connected to the structure of the database. These early databases were often called "navigational" databases, because they required a skilled navigator to traverse them. (Committee on Innovations in Computing and Communications: Lessons from History et al., 1999).

Pre-relational history is an important component of understanding the immense epistemological shift that relational modeling created. The relational database paradigm was not the first, or, for a while, even the most popular way of structuring databases. Before the innovations of the "database revolution," there were two popular types of databases. These databases, hierarchical and network, were rigid and highly structured, requiring that data be entered and connected following preset patterns. Using them required a sophisticated understanding of their operations, and of the data they contained.

Figure 6

A hierarchical database diagram.

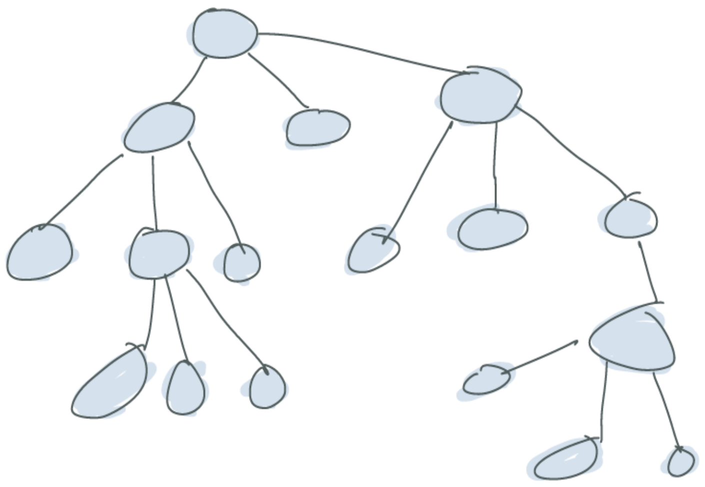
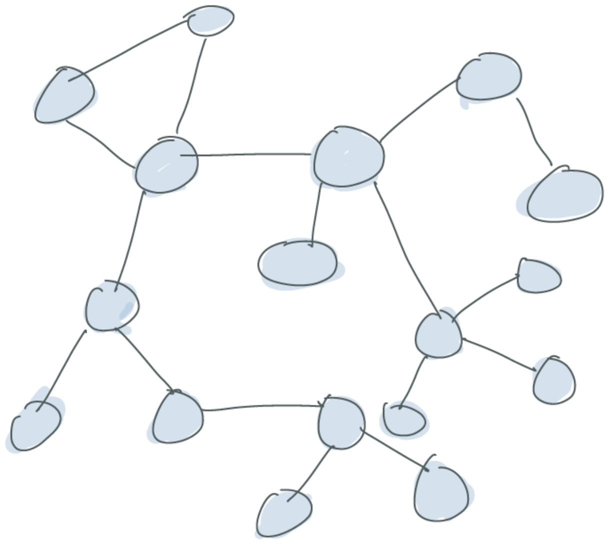


Figure 7

A network database diagram.



In the early 1970s, if you wanted to store data in a database, you essentially had two choices. In 1968, IBM introduced IMS - a database system that used a hierarchical model of data (Committee on Innovations in Computing and Communications: Lessons from History et al., 1999, p. 162). IMS stored data in a way that is easy to visualize like a family tree or organizational chart (Figure 6) Each item could be the child of another item, a parent of many others, and a sibling to others. The hierarchical structure was an innovation from no structure at all but had many major shortcomings for programmers: in order to use the structure, programmers needed to be familiar with the existing structure (i.e., one could not simply insert data anywhere); a significant change to how data was related might trigger a restructuring of the entire database; finding data (aka traversing the tree) was time-consuming and memory intensive. Importantly, and perhaps obviously, the hierarchical structure required data to be related in a parent-child relationship and forced developers to think of the data they wanted to store in this way.

In 1971, the Conference on Data Systems Languages (Codasyl) released a standard for database management (Committee on Innovations in Computing and Communications: Lessons from History et al., 1999, p. 162). This standard, often called the network model, allowed items to be linked to other items without having specific, pre-set rules about how many items might be linked to others (See Figure 7). In other words, when using the network model programmers were free from the parent-child constraints. The network paradigm resembled common internet/network diagrams---many nodes connected to many others. The network model was still quite rigid, however: data was required to be linked to other data, and many of those relations were artificial or inefficient; finding data still required traversing the tree in order to return results.

Using the network or hierarchical models did not require the database designer/creator to separate the structure of the database from its content (Dourish, 2014, p. 15). In practice, this meant that during the creation of the database there was no speculation about how the data would be structured. In a hierarchical database, the data would be stored hierarchically and the choices that were necessary were about which datum would be a parent and which a child. The content and the structure were the same—to a certain extent, the content dictated the structure and the structure dictated the content, both shaping where the next piece of data was stored. For a few years, database designers had the option to store data either in hierarchical or network configurations, and the data itself might indicate the best option.[34]

For example, data about genealogical records could be stored in a hierarchical structure fairly seamlessly, with parents naturally falling between grandparents and children. Data about flight paths might be stored in network databases, with hubs connected to other hubs.[35] In both of these early models, data were *interdependent.* Regardless of whether the relations between data felt artificial to programmers, data always had a type of context by way of these linkages. In both network and hierarchical structures, all data items were linked to other data, and the way items were stored in memory (aka, the "physical structure" of the data) matched the way items were visualized (aka, the "logical structure" of the data). As a result, there was no meaningful separation between the data's physical and logical representations. This is an important point because when programmers were using the network or hierarchical databases, there was no real need to model, visualize or diagram the data—the data and the database shared a structure determined by the technology. Every abstraction of a network database looked similar to Figure 7, every hierarchical database like Figure 8, but relational databases could look like nearly anything at all.

[34] Of course, there were a host of technical and financial considerations also at play here because different computer systems supported different models.
[35] The fit was not always so seamless. Data that might today seem more "flat," like bank transaction records, was often shoehorned into these structures

Data modeling

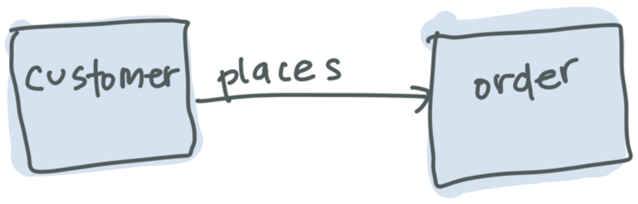
Edgar F. Codd introduced a new way of thinking about data and its "natural" structure that would offer new freedoms, and new epistemological and ontological frameworks to data thinking. Codd, a long-time IBM engineer, introduced the concept for the relational database model in a paper he published in 1970. "A Relational Model of Data for Large Shared Data Banks" opens with one of the fundamental objectives of relational databases—the separation between data's logical and physical structures: "Future users of large data banks must be protected from having to know how the data is organized in the machine" (Codd, 1970, p. 377). By their nature, databases of any form already separate data from their context, and with this move, Codd was working from what Tara McPherson calls lenticular logics, offering ways to separate data's physical structure from its logical: the relational model provides no dictates on the superstructure (i.e., tree, network) in which data are interrelated. In other words, the developer has complete discretion over how data are structured. The lenticular logic is a logic of the fragment or the chunk, a way of seeing the world as discrete modules or nodes, a mode that suppresses relation and context (McPherson, 2013).

The relational paradigm is built on two pillars: entities and their (eponymous) relationships. The smallest relational database might contain two entities with one relationship between them. In the very simplest terms, entities are nouns and relationships are how those nouns connect: a <u>customer</u> (entity) <u>places</u> (relationship) an <u>order</u> (entity).[36] This small database is diagrammed in Figure 8. Note that the entities <u>customer</u> and <u>order</u> are connected by a relationship.

[36] Relationships between entities are not always named in practice. However, in diagramming, its useful to label the interaction between two entities

Figure 8

A tiny relational database model



When introducing his relational model, Codd boasted that it "provides a means of describing data with its natural structure only---that is, without imposing any additional structure for machine representation purposes" (Codd, 1970, p. 377). The additional structures to which Codd is referring here are network or hierarchical models that forced data to be linked in ways that would be artificial, or to use Codd's terms, unnatural. What exactly *is* the "natural structure" of data? According to Codd, the natural structure of data is as entities in relationship to each other. In terms of how that data is stored, however, it means that relational models are structured around data collections stored as tables and related to each other in ways that are determined by the programming team, rather than required by the database structure itself. In other words, unlike hierarchical or network systems that controlled how data items were related (as nodes, or in parent-children-sibling relationships), relational databases allowed the programmers make relationship decisions and this allowance reinforces the separation between the data's "natural" structure and how it is stored in a system.

This new paradigm for thinking about data structures, and method for designing them amplified the power of data modeling within the software development lifecycle. The paradigm has persisted within the SDLC with little interrogation. Thus, the ER model becomes naturalized, eliding interrogation.

Because relational databases can take many forms, programmers were experimenting with new ways to visualize the data and the connections between entities. In 1976, Peter Pin-Shan Chen created the Entity-Relationship (ER) Diagram to provide a "unified view of data" (Chen, 1976). This unified view provided a visual vocabulary for several of the conventions of relational modeling. In Chen's ER Diagram work, he argued for diagramming data's "natural" structure, echoing Codd's use of the word "natural" a few years earlier. Chen writes "The entity relationship model adopts the *more natural view* that the real world consists of entities and relationships" (Chen, 1976, p. 9 emphasis in original). This is an immense epistemological and ontological claim that I address in more detail in the next section.

Even though Chen insists that data has a natural state, he acknowledges the inherent subjectiveness of the process when discussing relationships: "A relationship is an association among entities. For instance, 'father-son' is a relationship between two 'person' entities" (Chen, 1976, p. 10). However, as Chen notes, what becomes an entity is a "decision which has to be made by the enterprise administrator. He [sic] should define what are entities and what are relationships so that the distinction is suitable for his environment" (Chen, 1976, p. 10). With this statement, Chen positions the database designer (who he calls the "enterprise administrator"), as the individual with the power to determine what "things" deserve identification and attention, and thus which concepts are important and available.

Of course, there is no singularly "natural" state of data or a "natural" or "unified" view of the world. As we will see throughout, the use of the word "natural," is a powerful universalizing move designed to promote widespread adoption of relational databases and Chen's ER Diagrams. As I have argued elsewhere "by divorcing data from problems of storage with the goal of creating generalizable database recommendations, the relational model subtly reinforced the notion of data as 'natural' and corresponding to something prior to or separate from the ways it is collected, labeled, organized, and stored...Such naturalistic conceptions of data also separated them from the social, political, or economic contexts within which they are produced" (Stevens et al., 2021).

The epistemological commitments of modeling

Data modeling is a series of ontological and epistemological commitments. Within relational data modeling, the first ontological commitment is agreeing to view the world as a series of objects, properties of those objects, and relationships between objects (Olivé, 2007, p. 11).[37] Thinking in this way for long enough impacts one's perception of the world — this initial ontological commitment is very high stakes.

The relational ontological commitment is also an epistemological one—an implicit agreement that what is represented through those objects and relationships is true (or perhaps true enough for the system's purposes). It is our epistemological orientations that allow us to ask questions of databases, assuming that the data they return will be accurate. The commitment to trusting databases to provide knowable or usable data is also high stakes.

I argue that there is one especially key moment in terms of both significance and intervention: when an attribute becomes an entity. That is, when a modeler decides that something that is a property or descriptive field about an entity, itself becomes an entity. In data modeling, an entity is an object (person, place, thing, idea, or concept) that has attributes, or properties. Entities are related to other entities, in keeping with Chen's Entity Relationship Diagram. In databases which need to keep track of people, there is often a <u>user</u>

[37] Note that the word ontology has two different, though interrelated meanings. In branches of computer science, ontology is "a specification of a conceptualization" (e.g. deciding how to build a world full of objects and relationships) (Olivé, 2007, p. 11). Ontology is also a branch of philosophy dealing with the nature of reality. In this section, I am referring to the philosophical meaning

or customer entity. That entity might have attributes like email_address and first_name. The key moment, then, is when an attribute of an entity, becomes an entity itself. In this section, I describe the transformation from attribute to entity, and illustrate the ways that these choices are ones grounded in ideology and power. That data models represent such intense epistemo-ontological commitments is another reason they must be studied more carefully.

Figure 9

Marital status is an attribute of the user entity.

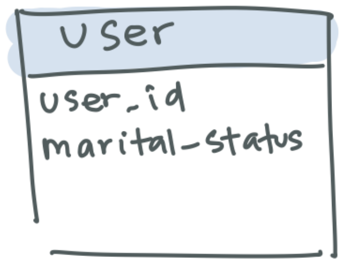


Figure 10

A user is linked to their marital partner.

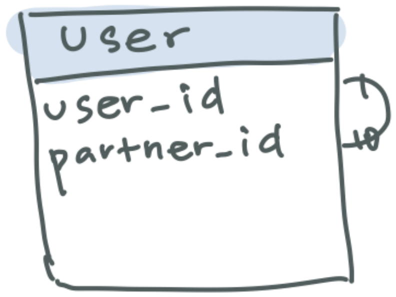
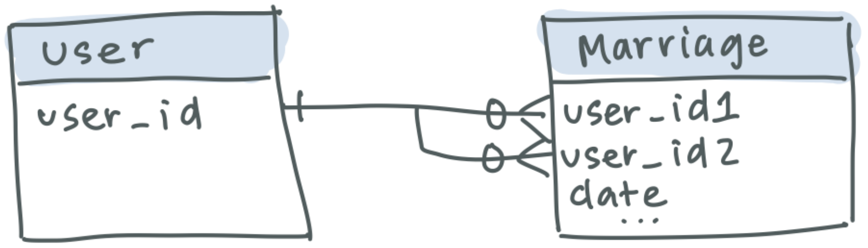


Figure 11

A user is linked to a marriage entity.



Deciding which questions, or type of questions, the data model needs to answer is part of the process of data modeling. Consider the difference between “Is person X married?”, “To whom is person X married?” and “Where and when did person X marry?” Each of these questions requires increasing amounts of detail about a person and their marriage. Figure 9 shows a User entity with an attribute marital_status enabling us to answer whether a person is married. The marital_status attribute might contain values like “Single, Divorced, Widowed, Married.” Figure 10 also shows a User entity, but this time the entity has a partner_id attribute. The line on the left side depicts a relationship, indicating that the partner is also a user in the table, and answering the question “To whom is partner X married?” Finally, consider Figure 11 where marriage is reified: the relationship between two User entities becomes an entity itself. When marriage becomes an entity of its own, those of us who are looking at this model are then free to think about other properties of marriages which we might want to store. In the example question above, I would opt to store the date, location, and perhaps officiant of the wedding ceremony. The facts of the marriage are the same regardless of whether it is modeled with Figure 8, Figure 9, or Figure 10. However, it is this crucial act of “promoting” marriage to an entity separate from, but related to, the user that facilitates thinking about other information to store.

As another example, consider a small, hyper-local context: a small business is making a list of customers to receive farm share deliveries during growing season. In this case, information about a delivery address might be most appropriate as an attribute to a <u>User</u> entity (see Figure 12). However, in an e-commerce context, customers might have several addresses stored and labeled for different transactions. Contrast Figure 12 and Figure 13. Note that in Figure 8 there is one user with an infinite number of addresses supported by this model---a user can have as many addresses as needed. Note the importance of the markings on the line connecting the <u>User</u> and <u>Address</u> entities. On the left, the vertical line through the horizontal line means one. The angled lines on the right mean many. Thus, the diagram is read: Each user can have many addresses. In this configuration, the primary key named user_uuid is the "foreign key" in the address table linking the address back to a specific user record. Those with data modeling experience might point out that splitting address from customer (i.e., making address its own entity) is a standard move because of the requirements of normalization. I use normalization techniques throughout this chapter and as a foundation for my own data modeling, but there is a growing movement advocating for ontological modeling. Ontological modeling is an emerging practice that "looks at the data and asks the question 'What does this represent?' and then it structures the data around that" (West, 2011, p. 65). Normalization is the process of looking at patterns and repetitions in the underlying structure of the data and modifying to eliminate them.[38]

[38] For a solid overview of normalization, see Date (2019), Simsion & Witt (2005).

Figure 12

A User entity with an address attribute.

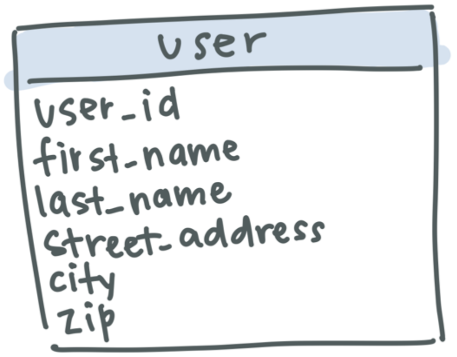
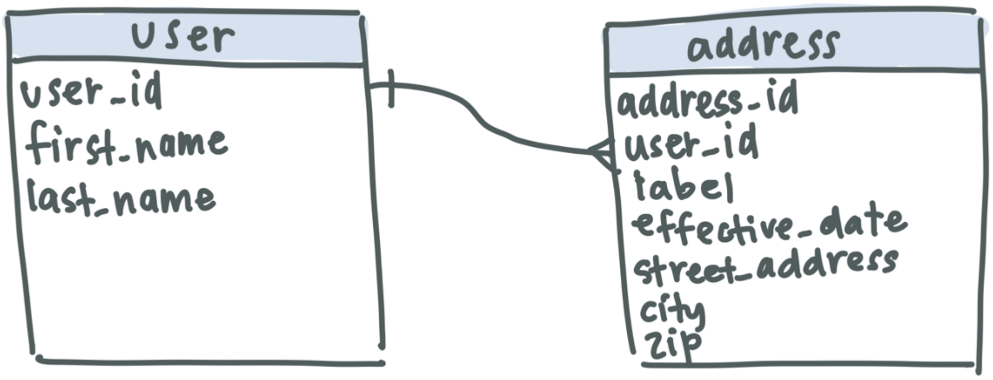


Figure 13

A User entity linked to an Address entity.



Normalization is a standard practice when creating relational data models. When a data model meets certain criteria, the model is said to be in a “normal form.” In 1970, E.F. Codd introduced the relational database paradigm, and over the next decade developed criteria for “normalizing” the database. While the intricacies of data model normalization are out of scope, a key point here is that through the process of normalization, some attributes become entities. The criteria by which that happens is based on the modeler's epistemological and ontological frameworks. In other words, it is customary to separate a

User from their addresses because there is a common understanding that people can have many addresses to which they ship packages. In both the marriage and address examples, promoting a field from an attribute to an entity facilitates more expansive thinking about the attribute-cum-entity. It is crucial to recognize that some attributes (address) are *de facto* promoted to entities and other attributes remain attributes and that choice is based solely on the epistemo-ontological frameworks of the modelers. In Chapter 4, I explore what happens when an attribute like gender is promoted to its own entity.

Throughout this discussion of the epistemological and ontological commitments of modeling, it is important to recognize that data modelers model data according to the requirements they have been given (as I described in Chapter 1). A modeling team may get a user requirement like "As a user, I want to have a profile with all of my information so that I can receive personalized product recommendations."[39] The requirement may specify which information about a user is stored ("name, email address, mailing address, age, favorite sports"), but it is at the *sole discretion* of the data modeling and database teams how that information is stored. In other words, the models are created to align with the modelers' views of the "natural structure" of data. Perceptions about that natural structure are inevitably influenced and determined by the social and cultural context in which the modelers are working. In the United States, that context is the infrastructure of white-cis supremacy.

As a specific example, gender is often stored as an M/F binary or M/F/X ternary. The most common practice of storing gender has been a "state" paradigm---which is to say,

[39] Remember that user story requirements are not necessarily "true" things that a user might want. They are created by the planning team after discussions with stakeholders. User stories about how a user wants personalized algorithmic recommendations are aspirational, based on product or monetization goals.

that the database records the active "state" of a value or setting in a system. When storing the state of something, there is a fundamental idea that there can be only one valid state in any given moment and that the number of states is *finite*. In the case of gender, people have a gender that is only one of a set number of possibilities. Initially, valid values for the "state" of gender would have been M and F. More recently, especially in commercial software applications and web-based forms, these valid states have been expanded to include values like "non-binary" and "MtF" or "FtM".[40] The addition of MtF and FtM values further imply a sense of statefulness, which is a generic computer science description of systems designed to remember preceding events or user interactions. Additions of gender fields to include additional values, a free-entry text box, or radio buttons which support the selection of more than one value (i.e., respondents can choose "non-binary" and "woman") are still working within a state paradigm. These data models and interface choices assume that users are able and willing to fix themselves to a single gender and a single set of pronouns.

Importantly, the act of promoting an attribute to an entity may seem solely like an epistemological shift. The affordances of an entity allow the data modeler to open new ways of knowing---to ask new questions about (in our examples) the marriage and the addresses. However, the act is about more than asking new questions and shifting epistemologies. The act of promoting an attribute to an entity can signal an *ontological* shift, especially when that attribute or entity is about an aspect of a person's identity, in this case, a person's gender. Starting with our initial ontological commitment---that the world is made of objects and relations---choosing to store gender as an attribute reflects the idea that gender is simply a property of a person, not related to anything, nor influenced by anything external. In other

[40] "MtF" is an abbreviation signaling that a person was coercively assigned male at birth and is a woman: "male to female." Similarly, "FtM" is "female to male" for those coercively assigned female at birth who are men.

words, people have only one gender and there is no additional information to be gathered about it. The relegation of gender to attribute of a person and not its own entity reflects a collective, historical inability to view gender as something about which to store information (in the case of the marriage example) or as something that a person might have more than one of (in the case of the address example). I discuss this example in more detail in Chapter 4.

Data modeling and white-cis infrastructure

Recall Tara McPherson's work on the racialized logics present in the UNIX operating system. The illusion of wholeness McPherson describes, perhaps rendered through a "unified view of data" obscures the ways that the relational paradigm extends the modular logics, creating increasingly fragmented, yet superficially connected, depictions of the world. These are, to use McPherson's term, "lenticular logics"— logics of fragmentation and "a way of seeing the world as discrete modules or nodes" (McPherson, 2018, p. 25).

In this section, I discuss several techniques of data modeling that enable data models to reinforce social power. Both were introduced by John Smith and Diane C.P. Smith in 1977. First, I discuss reification, the act of turning a relationship into a noun---cementing or fixing the connection between entities into something that has the tenor of fact. This mirrors the ways that whiteness both reifies and is reified in the United States. Second, I discuss generalization. Generalization is a bottom-up approach that erases the differences between instances (people, things, events) so that they can be stored together as an entity. For example, it is the principle of generalization that often requires Hispanic/Latinx people to choose "White" as an ethnicity category. It is a technical application of the way that both cisness and whiteness flattens difference to maintain social orders.

Reification

"[Reification] refers to an abstraction in which a relationship between objects is regarded as a higher-level object" (Smith & Smith, 1977, p. 106). For example, a relationship between two people (i.e., being spouses) can get reified into a "marriage" object for storage in a database or visualization in a model. Reification comes from linguistics, in which the idea of "nominalization" is one in which a verb (move) is viewed as a noun (movement). In technology, it is the transformation of a relationship (a verb) into an entity (a noun): to reserve becomes a reservation, to marry becomes a marriage. Reification is performed when it is not sufficient to simply note that a relationship exists.

For example, consider Figure 5, a diagram in which two users are linked as spouses. Contrast that to Figure 6, in which the relationship is reified and a marriage table added.

Reification is not an abstraction without implications. In his work on data modeling, Antoni Olivé notes that "[t]he act of reification does not add any new knowledge to a schema, but it is necessary when we need to record additional facts regarding the instances of a relationship type" (Olivé, 2007, p. 124). For example, if the system needed to describe not just that two people were married, but the date, location, officiant of that marriage, it would make sense to reify the marriage into an entity itself. Here Olivé's claim seems to naturalize and lower the stakes for the act of reification, asserting that it does not add new knowledge, but simply creates opportunities to store facts. Instead, I would argue that reification creates opportunities to *create* facts, and to create objectivity and fixity around a relationship between entities. Furthermore, as I discuss next, the act of reification is a key action of white-cisness.

In "Whiteness as Property," Cheryl Harris explores reification as a mechanism by which whiteness maintains its power — "expectations of white privilege" are transformed from living relational negotiations to objects (Harris, 2006, p. 1784). She references Georg Lukacs's description of reification, important enough here to quote in full:

> [Reification's] basis is that a relation between people takes on the character of a thing and thus acquires a 'phantom objectivity,' an autonomy that seems so strictly rational and all-embracing as to conceal every trace of its fundamental nature: the relation between people. (Lukacs in Harris, 2006, p. 1730)

The autonomy and phantom objectivity here is crucial. When, for example, a domestic-legal relation becomes reified into a two-person “marriage” entity, it becomes the default and natural way to relate two individuals and, by virtue of the data model, excludes or marks “inconvenient” those who need to be related in ways like domestic partnerships or arrangements between three or more people.

That reification has become a database modeling standard is counter to any assertions about data's “natural” state. As Chen notes: “It is possible that some people may view something (e.g., marriage) as an entity while other people may view it as a relationship. We think that this is a decision which has to be made by the enterprise administrator. [They] should define what are entities and what are relationships so that the distinction is suitable for [their] environment.” (Chen, 1976) The “natural” view of data, coupled with the contradictory, yet unremarkable idea that the administrator gets to decide how to model the data is reminiscent of a (false) “view from nowhere” that simultaneously treats their view as “natural” and imbues them with the power to interpret it flexibly. While this “interpretive flexibility” is a key part of the way that scientific-technical knowledge is constructed (Pinch & Bijker, 1984), the positioning of the administrator as the unquestioned expert on what “counts” for modeling is a configuration that feminist science and technology studies scholars have been pointing to as problematic for generations.

It should be self-evident how to record items when they exist in their "natural" state. Troy Duster calls this the "the fallacy of reification" which he describes as "the tendency to assume that categories of thought coincide with the obdurate character of the empirical world" (Duster, 2005, p. 1050). Similarly, Harris notes that "the law constructed 'whiteness' as an objective fact, although in reality it is an ideological proposition imposed through subordination" (Harris, 2006, p. 1730). It is this *mechanism* by which aspects of data modeling are able to support white-cisness. Indeed, reification was used "to identify and denounce the transformation of historical processes into ahistorical entities, human actions into things that seemed part of an immutable 'second nature.'" (Roediger & others, 1999).

When a database designer practices reification, they transform active, context-rich, situated actions into "ahistorical entities" that possess "phantom objectivity." Why does the person creating the database get to decide what is an entity and what is a relationship, especially in a context in which this structure is "natural" for data? The answer, of course, is that there is no natural structure for data. Only an ideological fallacy that enables the frictionless translation of the lived world into entities and attributes.

Generalization

Generalization is the second of Smith & Smith's abstractions. Generalization is "an abstraction in which a set of similar objects is regarded as a generic object...In making such an abstraction, many individual differences between objects may be ignored" (Smith & Smith, 1977, p. 106). They do not elaborate on the "similarity" in their definition, but their examples imply that it is a similarity of convenience that is at work. Considerations of similarity begin when developers are defining entities. Contrary to my own simplistic definition earlier, an entity is not simply a metonym for the grammatical term "noun."

Instead, entities represent aspects of the world that need to be stored. Modelers must be aware of aspects of the world they are attempting to represent.

Olivé but takes care to distinguish between a concept and an object (which the idea of "entity" conflates). A concept is "an abstract or generic idea generalized from particular instances," those instances being "objects" (Olivé, 2007, p. 12). The distinction between concepts and objects is an important one, related the computer science concept of instantiation—an object or thing that is an instance of a concept. In object-oriented programming, and broader object-oriented paradigms (of which relational databases is one), classes or concepts exist as "blueprints" for objects.[41] The car parked in front of my house is an instance of a concept that could be called "cars" or could more specifically be called "SUVs." This object-oriented paradigm requires that programmers/designers have access to concepts so that that they can recognize the "things" or "objects" belonging to that concept. If it is true that "[modelers] do not show a practical interest in concepts without instances" (Olivé, 2007, p. 13). What does this mean for instances belonging to concepts which modelers do not recognize? In other words, on what basis do modelers determine which concepts do not have instances? How do modelers decide how to generalize? Like with reification, data models are generalized based on the concepts to which modelers have access and in accordance with the erasures normalized by white-cisness.

When performing generalization in data modeling, it is considered a bottom-up approach: an attempt to make many disparate things fit in higher-level container. Modelers

[41] The introduction of programming languages (specifically Simula) that supported objects, and classes allowed programmers to develop technologies for generalization and abstraction. For a situated history of Simula, see Holmevik (1994). For a review of the roots object-oriented programming, see Dahl (2002). With Simula, the first object-oriented programming language, programmers had hands-on experience working with tools to support concepts like generalization. This is important for the development and evolution of programming technology, but has special relevance here.

perform generalization when they decide that an entity named Faculty and another named Student can be stored in an entity called Person. The combination of faculty and student into person ignores all the differences between them.[42]

A common objection to this might be something like "how would we store different objects in tables if we didn't erase some difference between them? If we did not decide which aspects 'matter' and which 'do not matter'?" We cannot, of course, turn the map into the thing (or in this case, the model). The differences that we choose to erase (or prioritize) are based on our specific cultural and epistemological world views and related to our access to the parent concepts (or superclasses) for the objects we are considering.

"While 'generalization' in UNIX has specific meanings, we might also see at work here the basic contours of a lenticular approach to the world, an approach which separates object from context, cause from effect." (McPherson, 2013, p. 27). Generalization—the removal of difference between objects (subjects) into a meaningful category—is a key move performed by white-cisness. For example, over time, the generalization (or erasure of difference) enabled a variety of people to be considered "white" in the United States. As Nell Irvin Painter (2011) describes, the category of "white" expanded several times, first to include Irish immigrants, later Jews and Italians. Differences that seemed worth of separate categories became erased and deprioritized to support a changing social order.

Throughout this chapter I have argued that data models are an important site of analysis, one that reflects onto-epistemological commitments. These commitments are not

[42] In a relational data model, there is often a Person entity with primary/foreign key relationships to entities about that person's specific roles. In the above example, someone might have an entry in both Faculty and Student tables linked back to their person entry.

simply those of the creators, but of the milieu in which the models are created. The relational paradigm requires modelers to view the world as a series of entities and relationships, a way of looking at the world and data that is more “natural” than the database structures that were available in the late 1960s. However, this “natural” view of data is only maintained through data modeling techniques that reinforce social systems of power. As a result, looking closely at the techniques of reification and generalization reveal that they are techniques that mirror some of the “interpretive flexibility” that white-cisness provides. Reification creates false objectivity in contexts that would be better served to remain messy and entangled. Generalization erases rich difference and flattens the world to reduce complexity. Data models, then, support systems of power and demand an extensive amount of investigation, especially in contemporary software applications.

CHAPTER 3

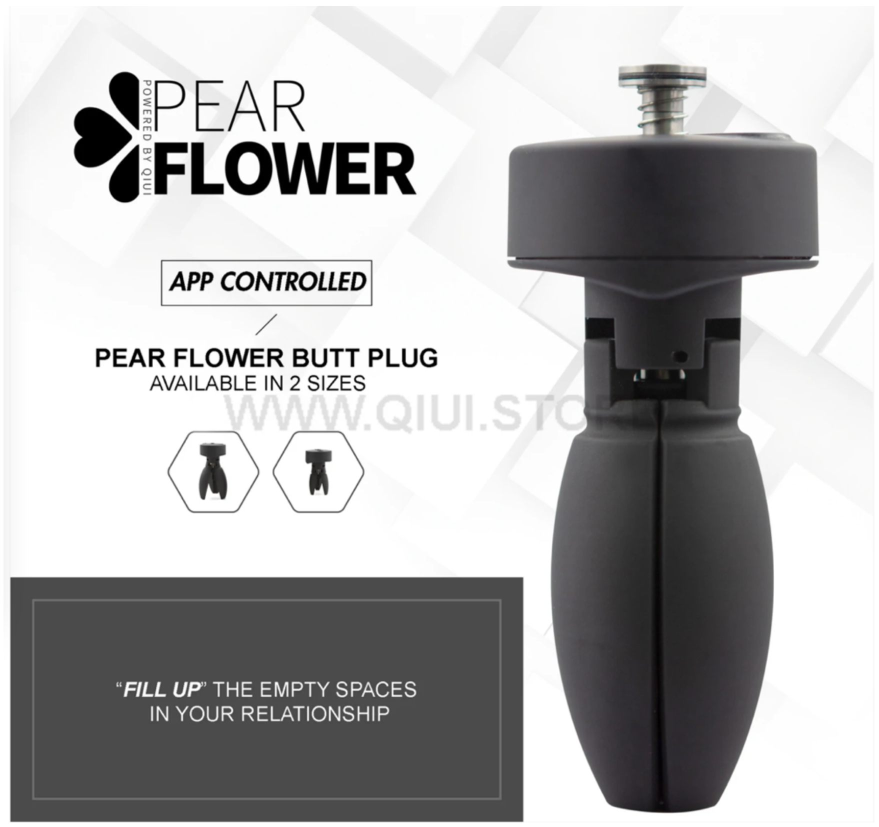
LOOKING AT DATA MODELS

When data models become databases powering software applications, the impacts of data modeling choices become encoded in the software. Unfortunately, it is rarely possible to obtain the original data model used to develop a particular software application, which is why, in this chapter, I use reverse engineering to infer the data models used. In this chapter, I demonstrate a methodology for obtaining the data models and, rather than extensively draw my own conclusions because that is a project in itself, show how to think about the models in their analytic contexts. This is a foundational offering to scholarship across disciplines that investigates the social impacts of software.

When considering the impacts of software, I am particularly interested in software that has impacts for individuals in their homes. As I discussed in the introduction, surveillance in the United States is racialized and gendered. Thus, I open this chapter with an example of the immediate harms that software and hardware can cause to an individual, and then proceed to uncover the data models within Lovense software.   It is crucial for those of us who study software to look beyond their external results. Opening software to discover the data models within is an important way to examine the manifestations of power within data modeling, and software more broadly.

Figure 14

An image for the Pear Flower, an internet-connected sex toy made by Qiui.



"Fill up the empty spaces in your relationship" is the tagline for the "Pear Flower," a butt plug made by Chinese company Qiui (Figure 14). However, the Pear Flower is distinct from other anal play devices. It is controlled by a mobile app and with simple a tap of the app screen, the device "blooms" in the wearer's rectal cavity, locking it in place. The Pear Flower is one of many devices designed for adults engaging in dominance and submission sexual play (a part of the BDSM umbrella).[43] Qiui makes several of these devices, including

[43] For an introduction to critical literature on BDSM, please see the BDSM special issue of *Sexualities*. (Simula, 2021).

an electric shock collar (the "Little Devil"), and a chastity cage (the "Cellmate"). Similar to the Pear Flower, Qiui's Cellmate locks a user-wearer's penis in a chastity cage that the user-wearer cannot open manually. "QIUI believes that a true chastity experience is one that does not allow the wearer to have any control over" (Qiui, n.d.). Access to the cage is controlled by a cellphone application that connects to the device via Bluetooth.

The conceit of Qiui's Cellmate and Pear Flower is that for some people abdication of control is a key element of sexual pleasure. The idea is not new, of course, but traditional locking sexual devices are usually un/locked with a common lock and key; the wearer gives the key, and thus control, to their partner. If a key is lost, its lock can often be picked with a bobby pin or other common lock-picking device. Qiui products simplify the need to track and manage keys, allowing one user account to control many devices, locking and unlocking as the keymaster desires, with no need to even be in the same room as the wearer(s).

What happens when you connect sexual devices to the internet? They get hacked. In 2020, a security testing group released a report revealing that because of a security flaw in part of the application's code, anyone could seize control of a Cellmate (Pen Test Partners, n.d.). Without access control, and because of its strong locks, extrication from a locked Cellmate would require bolt cutters or an angle grinder very close to the user-wearer's penis. This is probably not the kind of submission the participants intended when they opted into this device.

The Cellmate is only the most recent internet-connected sex, or teledildonics, device to have security flaws.[44,45] The smart vibrator Vibratissimo had a similar flaw that let anyone

[44] See Wynn (2018) for a review of the privacy and security of several internet-connected sex devices
[45] Teldildonics (sometimes also called "cyberdildonics") is used to mean "sexual interaction mediated through remote control sex devices"(Faustino, 2018, p. 2). Although the term "dildonics" was coined in the early 70s

gain control over the device. However, Vibratissimo's device security flaws were matched with data security flaws – "Usernames, plaintext passwords, chat histories, and explicit image galleries that users created themselves were sitting in an" unsecured database (Whittaker, n.d.). In 2018, a woman sued the makers of the Lovense sex toy for collecting data about her use of the product, mirroring a similar suit against the makers of a sex toy called We-Vibe in 2017. Both cases were settled out of court (*N.P. v Standard Innovation Corp*, 2017).[46]

For users of internet-connected sex toys, these teledildonics products likely feel like simple pleasure between consenting partners—a fun new way to connect, or a way to stay sexually connected over distance. Unfortunately, teledildonics devices, like other Internet of Things products, are not simply entertainment; they are also surveillance. IoT devices live in the home, which is a "prime data collection node" (McGuirk, 2015). While other works take up the impact that this *data* can have in individual lives, I want to take care not to become fixated on the content of the data itself---that this data has certain contexts or contours is a factor of the data model and database design, themselves factors of the cultural-business milieu in which they were designed. The milieu of IoT sex devices is one heavily influenced by US surveillant logics. As I have discussed, surveillance in the United States is largely a practice designed to uphold the infrastructure of white-cis supremacy.

In what follows, I describe in detail the process of reverse engineering the Lovense Android application and how, through looking at the code, I was able to reconstruct the data model, a process both highly technical and inherently speculative.

(by the same person who coined the word "hypertext", Ted Nelson), the "tele" was popularized by Howard Rheingold in 1991 (Faustino, 2018).

[46] For information about several large commercial data leaks, see Shaban (2018), Newman (2018), Glaser (2018). See Duhigg (2012) for an example of consumer behavior prediction. See Eubanks (2018) for information about how the state uses algorithms to determine the services a citizen may need. For a sampling of information on algorithmic bias, see O'Neil (2016), Chander (2016), Garcia (2016) Lodge and Mennicken (2017), Sandvig et al. (2016), Yapo and Weiss (2018).

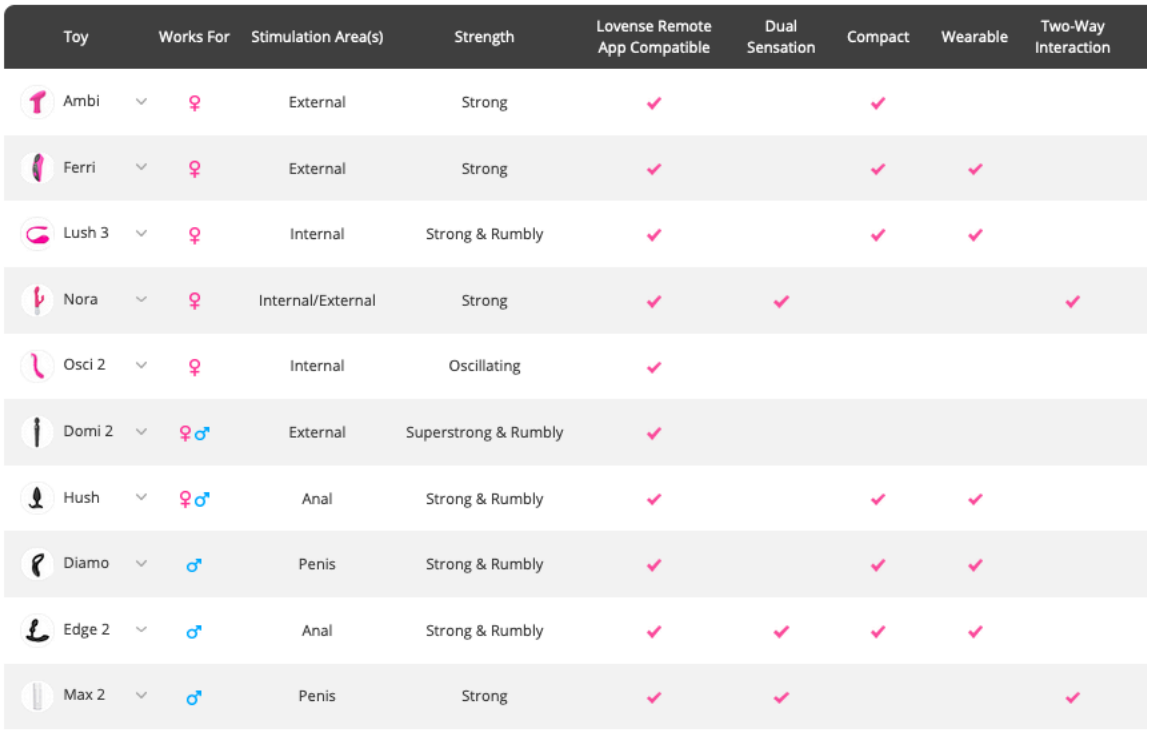
Lovense

In the next section I outline the "technical" steps included in reverse engineering. However, with this section I begin with a foundational context and information about Lovense, the product being investigated. While close reading has been described as "the quintessential humanist methodology," it is also by rights part of reverse engineering, in that it is "a detailed examination, deconstruction, and analysis of a media text" (Bizzocchi & Tanenbaum, 2011, p. 262). The close reading that follows is part of the reverse engineering of the device.

Lovense, Inc was founded in 2009, because "the founder of the company was in a long-distance relationship. Lack of sexual intimacy was a serious problem in the relationship and sparked an interest in teledildonics" (*Lovense Bluetooth Sex Toys - History of Innovative Sex Tech*, n.d.). Lovense makes "sex tech for everyone" --- a line of teldildonics products that boast "long-distance interactivity," "programmable vibration strengths," and "wireless remote control" (*Lovense Sex Toys for Long-Distance Love*, n.d.). The Lovense product line contains familiar sex toy forms, like rabbit vibrators, penis masturbaters, and anal plugs. Refer to Figure 15 for a table of their product offerings at the time of this writing.

Figure 15

A table listing Lovense product offerings.



Toy	Works For	Stimulation Area(s)	Strength	Lovense Remote App Compatible	Dual Sensation	Compact	Wearable	Two-Way Interaction
Ambi	♀	External	Strong	✓		✓		
Ferri	♀	External	Strong	✓		✓	✓	
Lush 3	♀	Internal	Strong & Rumbly	✓		✓	✓	
Nora	♀	Internal/External	Strong	✓	✓			✓
Osci 2	♀	Internal	Oscillating	✓				
Domi 2	♀♂	External	Superstrong & Rumbly	✓				
Hush	♀♂	Anal	Strong & Rumbly	✓		✓	✓	
Diamo	♂	Penis	Strong & Rumbly	✓		✓	✓	
Edge 2	♂	Anal	Strong & Rumbly	✓	✓	✓	✓	
Max 2	♂	Penis	Strong	✓	✓			✓

Looking at the Lovense product lines and feature offerings reveals a richly complicated view of gender, sexuality, disability, and sex work. Like most sex toys, Lovense products hew to a cissexist, and biologically essentialist gender binary. On their product table, they conflate particular genital configurations with gender identities. Note the "works for" category in the figure, which assumes that "men" (♂) are users of devices that stimulate penises and "women" (♀) are the audience for devices that stimulate "internal." Two of their products, the two I obtained for this project, Max and Nora, have traditionally Western gendered names. Max 2 is billed as "revolutionizing male masturbators!" (Lovense, n.d.)

and as a "high-tech male masterbator." In this language, they reinforce the gender-genital connection, assuming that everyone with a penis is male and that all men have penises.[47]

Their descriptions of Max reinforce heternormative ideas about what people with penises desire---presumably pink vaginas. Max has a plastic outer shell and a removable and washable silicone insert. By default, Max is shipped with an insert that is a translucent clear (nearly grey) material (Figure 16). The product's marketing simultaneously reinforces heterosexuality and whiteness by offering a "vagina sleeve" that is pictured as a light pink (Figure 17).[48] While it is out of scope to explore how the company's choice of language might impact the user's relationship to the people-with-vaginas in their own lives, it is worth noting that instead of advertising that the user is able to control the toy, they say that the user can "[c]ontrol the vagina contraction and vibration settings" (Lovense, n.d.). Nora's marketing copy fares a little better: they describe Nora as "[d]esigned to pleasure your G-spot!" (Nora by Lovense. The Original Bluetooth Rabbit Vibrator!, n.d.) and as the next evolution of "devices with a "Flexible Clit Arm." The branding page for Nora does not mention the word woman.

[47] There are masturbators made explicitly for trans men who have experienced "bottom growth" (aka clitoral enlargement) due to testosterone. They are typically called "FTM strokers."

[48] As some authors note, discussions of the operations of teledildonics, are naturally intertwined with descriptions of sexual activity and gendered bodies (Liberati, 2017). In this work, I operate from an explicitly transfeminist point of view and divorce all connections between an individual's gender and the configuration of their genitals or sexual preferences.

Figure 16

The clear liner for Max

Figure 17

The "vagina" liner for Max.

Option to purchase a Vagina Sleeve (at an additional cost) on our Store page!

Despite the cisnormative language on the website, Lovense has made a few non-normative choices in its products. First, in its operations (through code and how the application works), Lovense lets any two connectable toys "connect" to each other in a way that is unexpected given the website copy. In other words, two Maxes or two Noras can be paired and sync via the app—a move that is hardly as heteronormative as the copy might imply. Second, Lovense has made a long-term investment in making their devices accessible to cam workers. The device is designed to be easy to operate for individuals who perform "cam" work—a shorthand for people who charge money to viewers who watch them do (often, though not exclusively) sexual acts on video. The Lovense ecosystem was designed with this in mind. Third, the app enables "voice control" over Lovense toys. Commands like "Alexa, make me moan with Max" or "Alexa, please my partner" are available to Lovense customers. The integration provides accessibility to those who might have motor control issues. While itis facile to dismiss this integration as another manifestation of surveillance culture, looking beneath that we can see this integration as a move toward inclusion—as one of its *many* possible interpretations. I note this specifically as a way to resist superficial dismissals or assessments of a developer/product team's priorities and as a way to signal that this reverse engineering process is one of inferring, and not knowing explicitly.

Without the following technical steps of reverse engineering, any possible analysis of the Lovense product is limited to the website copy and the interface of the application itself. However, in this chapter I demonstrate a toolset that provides deeper layers of analysis. It also serves as proof that data models, while difficult to obtain, are not universally inaccessible. With the right steps, data models are available for examination and anlysis.

Uncovering the data model

In order to comprehensively analyze software in the context of its use, the data model must be part of the analytic landscape. Without the data model, analysis like a close reading of code, or assertions about algorithmic bias or dataset bias are incomplete, because it is the data model that creates those possibilities. However, obtaining the data model of commercial software is nearly impossible -- the data model is a series of artifacts that are created in the process of making software. The shape of the data model directly impacts the shape of the database used in the software, but the model itself remains elusive. Though it is rarely possible to see the original data models of production software, it is possible to infer the database schema (aka, the physical data model) and then to make inferences from that model. In this section, I offer a technical methodology for obtaining the physical data model and demonstrate how to infer the conceptual data model. As my use case, I continue with the Lovense product and move stepwise through reverse engineering the Lovense application.

Reverse engineering processes might seem abstract to those without experience making software. If we think of a piece of software as analogous to a house, one reverse engineering project might be to draw a set of architectural, plumbing, and electric schematics of an already-built house. To find support beams, wiring harnesses, and large pipes will require removing drywall and inspecting the contents between the walls. It might require digging up some of the front yard to figure out where the house connects to the municipal water supply. It is possible to do this reverse engineering without destroying the structure of the home, but it will almost certainly make the home unlivable for a time. To reverse engineer a house thoroughly requires more than a visual inspection, or even an understanding of how the house has been lived in; it certainly goes beyond comparing one house to other houses or looking at neighborhood zoning laws. In this way, reverse engineering processes help us to understand software in a way that we cannot by merely doing external analyses.[49] In this project, instead of a house, I had the Lovense Android application, called Lovense Connect.

Lovense has iOS and Android mobile applications. In cases like this, where applications exist for two operating systems, those applications will use the same database and data model. I opted to reverse engineer the Android application (aka, I chose the house with unlocked doors); reverse engineering iOS applications is significantly more difficult because iOS is closed source while Android is open-source software. Because Apple can control the hardware on which their software runs, they have fewer concerns about software-hardware compatibility and no need to use technologies designed for interoperability. Apple and all iOS/OSX devices operate within a closed-source technological ecosystem. That is, nearly all of their code or hardware specifications are held securely within their control and oversight, and they make available only what is needed to foster a profitable developer community according to their growth models. Created by Google, the Android operating system has always been open source. This means that the code that runs the operating system is available for download, contribution, modification, and remixing (*Android Open Source Project*, n.d.).

The Android platform, as well as most Android applications are built in Java, a decades-old programming language. When it was released in 1996, Java represented an important innovation. Until that time, software was hardware specific. Code was written and compiled for specific types of machines (recall the proliferation of branded software

[49] Please do not read this as a discrediting of existing analyses of software or their impacts. Instead, I am trying to offer ways to develop deeper understandings of software as cultural objects.

packages like IBM-specific office management tools). Java was one of the earliest languages that decoupled this dependence between software and hardware (code and machine). Programmers can write Java on their own machine and prepare binaries for distribution. Then, an operator can run that Java code in a Java virtual machine that adds system-specific libraries. This means that one Java program can run on Windows, OSX, and Linux because each of those machines will run the code inside a platform-specific container. This interoperability made Java a natural choice for a mobile operating system that needed to run on many different types of hardware. Because of Java's age and ubiquity, there is a robust reverse engineering toolkit for Java applications. In other words, lots of people have been working for a long time on easy ways to hack into Java apps.

While there is no perfect set of steps that every reverse engineering process follows, I divided this reverse engineering process into four steps.[50] First, I obtained, unpacked and investigated the Android software package itself. Next, I deobfuscated code and read the code for clues. Then, I developed a physical data model. Finally, I created a conceptual data model based on what I had learned from the previous steps. In this section, I offer a detailed description of each of these four steps of reverse engineering the Lovense Connect app. Readers less interested in the technical details can safely skip to the next section.

Breaking In

The first step is to download Lovense Connect to a device that will allow it to be manipulated—that is, to get the mobile application onto a laptop instead of a phone. Like all Android apps, Lovense Connect is packaged in an APK. An APK is simply a renamed zip

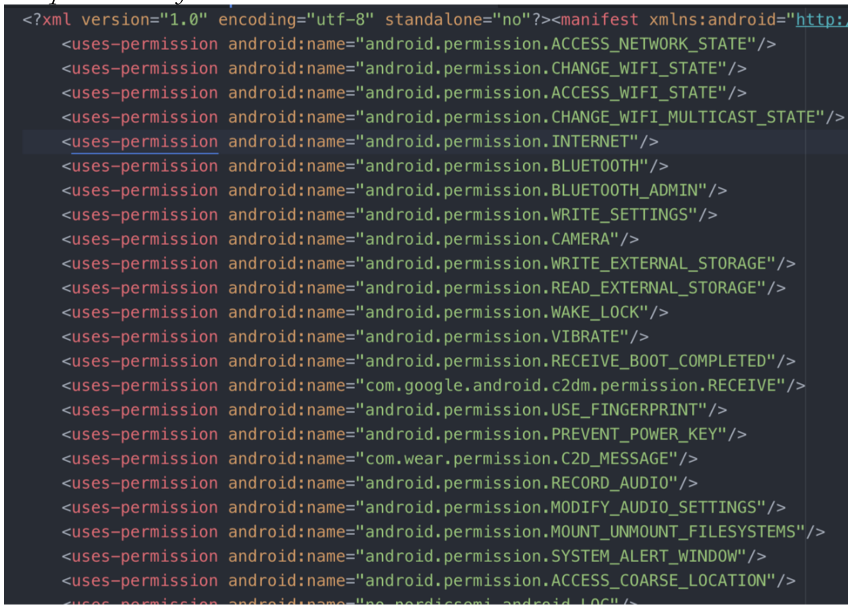
[50] For a variety of approaches to reverse engineering, see Baloch (2017), Canfora, Harman & Di Penta (2007), Z. Chen et al. (2019), Dang (2014), Dolan-Gavitt et al. (2015), Eilam (2005), Harris et al. (1995), Lanza & Ducasse (2003), Lin et al. (2010), Rekoff (1985).

file, which is itself a folder “zipped up” or compressed into a smaller file size. Typically, APKs are obtained via the Google Play store on an Android device. However, with my goal of putting the APK on my laptop, I used a mirror of the Play store that I could access in a web browser. Because the APK is essentially a zipped folder, it needs to be uncompressed. One simple way to do this is with software called apktool. Invoking apktool in a terminal will unpack the APK and allow inspection (Tumbleson & Wiśniewski, n.d.)

With only this first step complete, clues about the application emerge, including important operating system files, and the still-obfuscated code of the application. The results of this initial breaking in are artifacts worthy of analysis in their own right: the operating system files provide an orienting first look into the workings of Lovense Connect. The AndroidManifest is an xml file that contains information about the application, the permissions it uses, and its hardware and software requirements (*App Manifest Overview*, n.d.). Figure 18 shows an excerpt of the Lovense app's AndroidManifest.xml permissions section. The file is a key guide to navigating the rest of the application. Additionally, all APK folders contain a resources folder, “additional files and static content that [the] code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more” (*App Resources Overview | Android Developers*, n.d.). Browsing the resources folder is like looking at pieces of the interface—images, text—disconnected from their context and use. The strings.xml file (pictured in Figure 19) creates a variable for each piece of text (string) used in the interface and provides the display text. There is typically one strings.xml file for each language into which the app is translated. Looking carefully through each of these files helps to get a 10,000-foot view of the application, and although data models are not yet available, there is already more information to study.

Figure 18

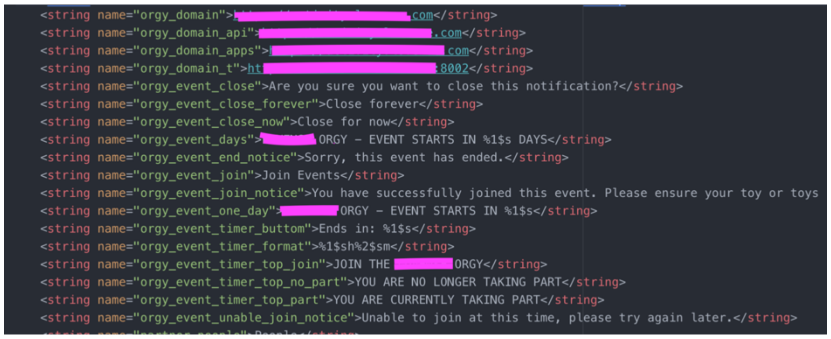
The permissions file in the Lovense APK.



```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http:
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
    <uses-permission android:name="android.permission.PREVENT_POWER_KEY"/>
    <uses-permission android:name="com.wear.permission.C2D_MESSAGE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Figure 19

The strings.xml file for the Lovense APK.



```
    <string name="orgy_domain">[redacted].com</string>
    <string name="orgy_domain_api">[redacted].com</string>
    <string name="orgy_domain_apps">[redacted].com</string>
    <string name="orgy_domain_t">ht[redacted]:8002</string>
    <string name="orgy_event_close">Are you sure you want to close this notification?</string>
    <string name="orgy_event_close_forever">Close forever</string>
    <string name="orgy_event_close_now">Close for now</string>
    <string name="orgy_event_days">[redacted] ORGY - EVENT STARTS IN %1$s DAYS</string>
    <string name="orgy_event_end_notice">Sorry, this event has ended.</string>
    <string name="orgy_event_join">Join Events</string>
    <string name="orgy_event_join_notice">You have successfully joined this event. Please ensure your toy or toys
    <string name="orgy_event_one_day">[redacted]ORGY - EVENT STARTS IN %1$s</string>
    <string name="orgy_event_timer_buttom">Ends in: %1$s</string>
    <string name="orgy_event_timer_format">%1$sh%2$sm</string>
    <string name="orgy_event_timer_top_join">JOIN THE [redacted]ORGY</string>
    <string name="orgy_event_timer_top_no_part">YOU ARE NO LONGER TAKING PART</string>
    <string name="orgy_event_timer_top_part">YOU ARE CURRENTLY TAKING PART</string>
    <string name="orgy_event_unable_join_notice">Unable to join at this time, please try again later.</string>
```

Creating code, reading code

At the end of the first step, the apk is uncompressed and each individual file is visible. However, the text within each file is unreadable; that is, each file contains series of seemingly scrambled alphanumeric characters that are in an assembly language called Smali.[51]

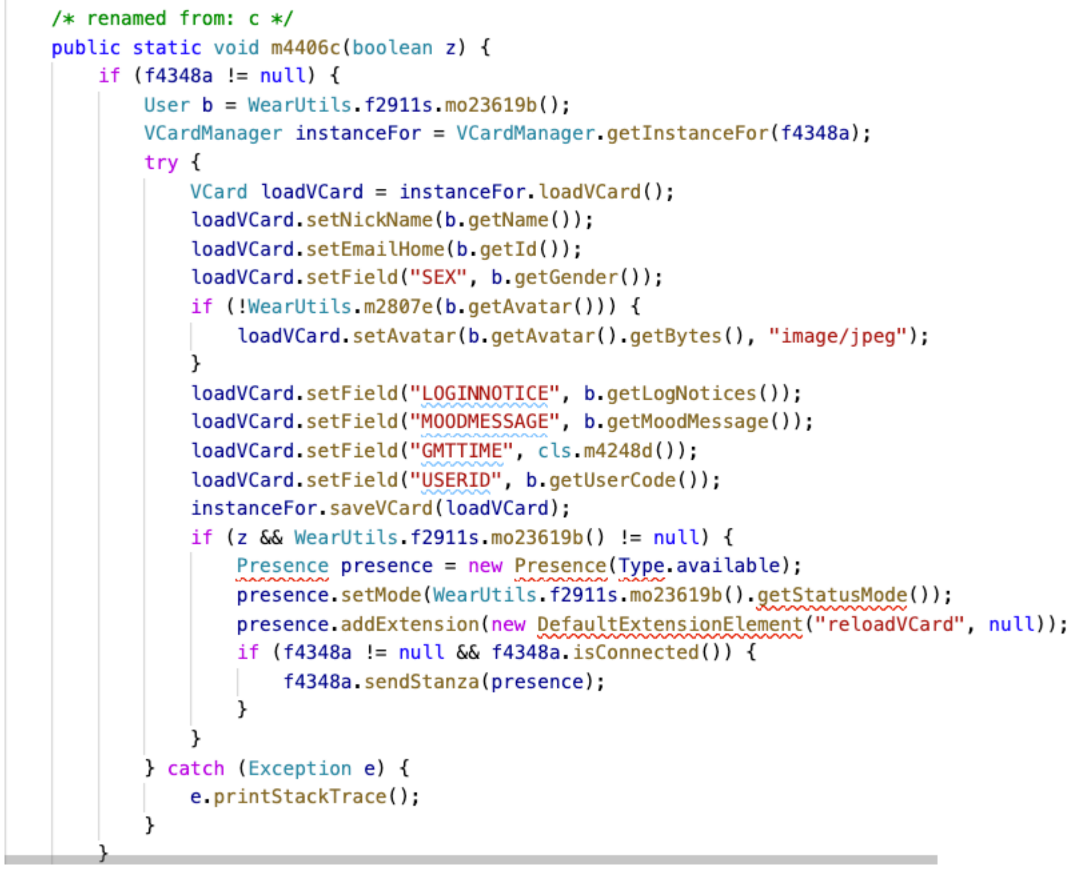
[51] Assembly languages are considered to be "closer" to machine languages and "farther away" from the incredibly human-intelligible languages that most of us write code in today.

The code for the application was not written in Smali: recall that code for this application was written in Java. As part of the packaging process, the Java code is compiled into Dalvik bytecode (or .dex files)---this is the format that Android OS uses. However, bytecode is even less human readable. As a help, apktool converts this bytecode to an assembly language called Smali. These many conversions are exemplary of the multiple mediations required to turn "code" into "software" and the mediations required to turn software back into code.

A folder of Smali code is not ideal for our purposes. The second step is to convert the Smali code to Java source code. Like all others, this mediation does not yield in a perfect translation. Using a Java decompiler, .dex files can be imperfectly translated into Java. The result of this decompilation is a series of folders and Java files. If the code has not been deliberately obfuscated by the author(s), it will be very close to the state it was when it was packaged to be delivered to the Google Play store. Deliberate obfuscation results in variables, function/method signatures, and class names being converted to semi-random strings like a3f3 that allow the code to execute but not be intelligible. After deobfuscating the Lovense Connect code, some files were completely readable with intelligible variable names (like emailaddress or logintime). Others resembled Figure 20 where parts of the code were intelligible and other parts obfuscated.

Figure 20

Partially obfuscated code. Note some variable names are random strings.



```
/* renamed from: c */
public static void m4406c(boolean z) {
    if (f4348a != null) {
        User b = WearUtils.f2911s.mo23619b();
        VCardManager instanceFor = VCardManager.getInstanceFor(f4348a);
        try {
            VCard loadVCard = instanceFor.loadVCard();
            loadVCard.setNickName(b.getName());
            loadVCard.setEmailHome(b.getId());
            loadVCard.setField("SEX", b.getGender());
            if (!WearUtils.m2807e(b.getAvatar())) {
                loadVCard.setAvatar(b.getAvatar().getBytes(), "image/jpeg");
            }
            loadVCard.setField("LOGINNOTICE", b.getLogNotices());
            loadVCard.setField("MOODMESSAGE", b.getMoodMessage());
            loadVCard.setField("GMTTIME", cls.m4248d());
            loadVCard.setField("USERID", b.getUserCode());
            instanceFor.saveVCard(loadVCard);
            if (z && WearUtils.f2911s.mo23619b() != null) {
                Presence presence = new Presence(Type.available);
                presence.setMode(WearUtils.f2911s.mo23619b().getStatusMode());
                presence.addExtension(new DefaultExtensionElement("reloadVCard", null));
                if (f4348a != null && f4348a.isConnected()) {
                    f4348a.sendStanza(presence);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

With mostly readable code, it is time to read the code to investigate the code's connection to its data source.[52] This is the first move towards uncovering the app's data model. Java applications like Lovense Connect can use several different techniques to create and interact with databases. In some cases, connections to the database need to be inferred from lines of code that directly read or write data. Fortunately, Lovense Connect uses in-code annotations; that is, the code describes the database tables and fields it is expecting. Annotations are a way of describing the physical design of a database and specifically the

[52] My reading of code here is a simple reading. Those interested in more complex and critical readings of code should refer to Marino (2020).

way that classes (or, instantiated Java objects) should persist into a database. Finding annotations is simply a matter of searching the codebase for common words used in these annotations. For example, inside one of the app's files is the following code:

```
@DatabaseTable(tableName = “tb_back_work”)
public class BackWork extends BaseEntity {
   @DatabaseField
   private String owner
   @DatabaseField
   private String staticParams;
   @DatabaseField
   private String status;
   @DatabaseField
   private String targetEmail;
   @DatabaseField
   private String workId;
```

All annotations follow a prescribed format determined by programming language standards.[53] The format requires that the annotation specify the name of the table, and the name of each field in the table. Reading the above, an excerpt from a file named BackWork, reveals that the code is expecting/creating a table named tb_back_work with five string fields: owner, staticParams, status, targetEmail and workId. Discovering that the project uses annotations and that they seem to be mostly deobfuscated means that the next step is to locate all of these annotations in the codebase and gather them into one file.

Let’s get physical

Recall that data models exist at three levels -- conceptual, logical and physical. The conceptual data model lays out all of the important things (aka, entities or objects) that an application and database might track. The logical data model adds detail to the conceptual, indicating the relationships between those objects and perhaps the flow of information.

[53] For more information, see https://ormlite.com/javadoc/ormlite-core/com/j256/ormlite/table/DatabaseTable.html

Finally, at the physical level, the data model dictates the details of database tables, fields, and field types.[54]

While traditionally a conceptual model precedes a physical model, in this case we start with the physical.[55] In order to do this, I transformed the file of annotations into a series of SQL CREATE TABLE commands.[56] CREATE TABLE statements are commands in the SQL programming languages that do as their name implies -- they create complete tables within the active database. The CREATE TABLE statement for the annotation discussed above looks like this:[57]

```
CREATE TABLE IF NOT EXISTS `tb_back_work` (
  `owner` TEXT NULL,
  `staticParams` TEXT NULL,
  `status` TEXT NULL,
  `targetEmail` TEXT NULL,
  `workId` TEXT NULL)
        ENGINE = InnoDB
        DEFAULT CHARSET = utf8mb4
        COLLATE = utf8mb4_general_ci;
```

Executing all of these create table statements on a mysql database server would yield a working database that, to the best of the code inferences, supports Lovense functionality. However, a visual representation of this database is more useful. I used a data modeling program to visualize each of the database tables and to begin to uncover how the tables were connected through primary-key foreign-key relationships based on the field names. In other words, the annotations alone do not reveal that the owner field above is the same as a database-wide user account identifier. I had to determine those relationships on my own.

[54] A very detailed physical model would include all primary/foreign key relationships, length and validation restrictions for fields; in other words, everything required to create the database.

[55] There are other ways to extract a database schema. See Andersson (1994) and Lubis & Zamzami (2020) for two examples.

[56] To do this, I used the command-line tool {grep} to search the codebase and pipe the results into one text file. I then used a set of regular expressions to transform the annotation format into an SQL format. With these steps, I was often able to get complete table contents.

[57] The complete SQL creation script that I used to create the initial physical model is available in Appendix A.

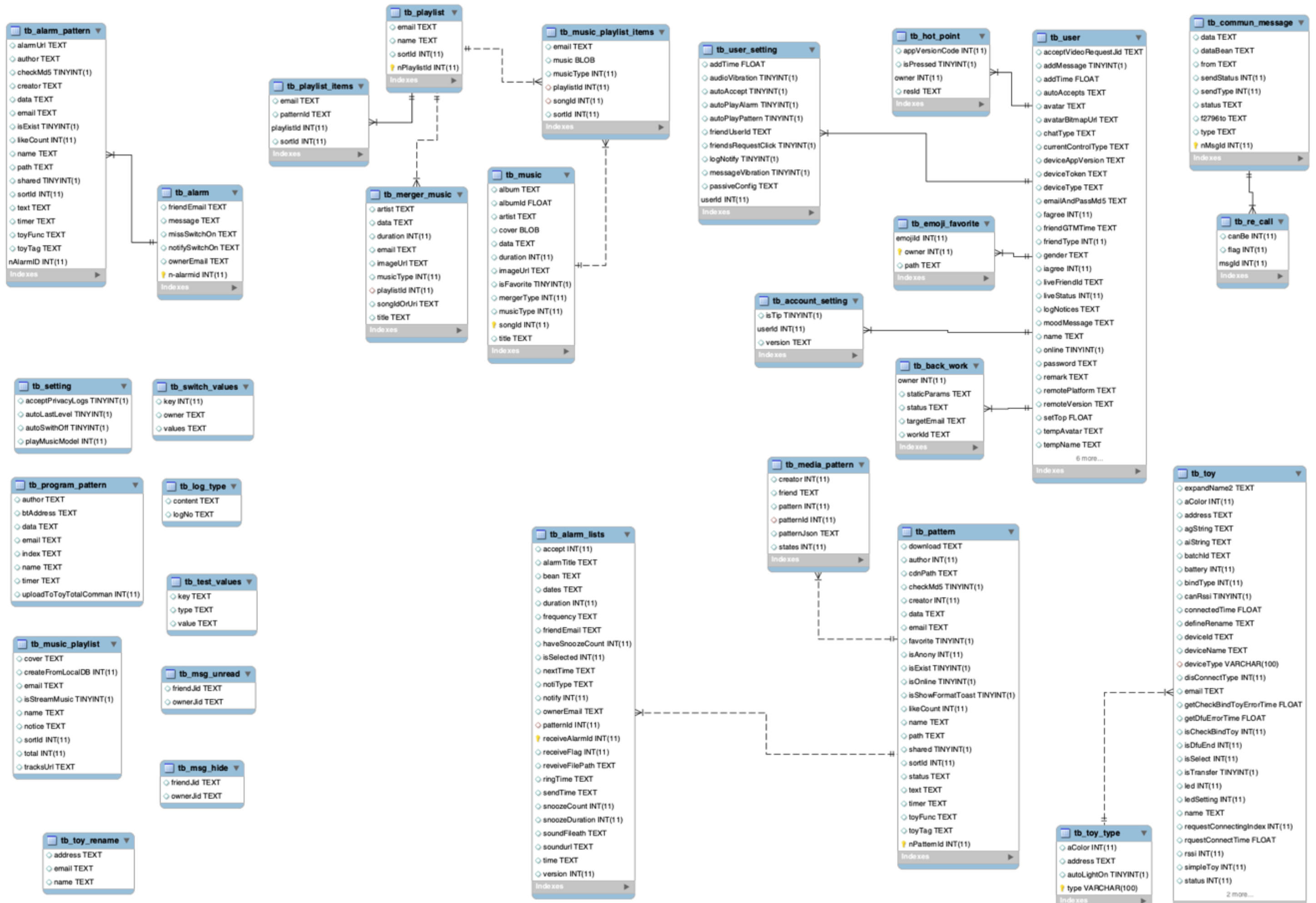
Like data modeling itself, this was a bit of a creative process. In some cases, the relationships were self-evident. For example, a table named user might have a field named id, and another table might have a field named user_id. Based on my experience with software engineering, I read through the tables, creating links and making a physical model based on both the code annotations and common relational best practices (see Figure 21).[58]

Constructing the physical data model (Figure 21) was not a straightforward operation for several reasons. First, while the model reflects the annotations in the codebase, the Lovense APK's codebase was partially obfuscated, and contains what I believe to be some "legacy" code.[59] The partial obfuscation and legacy code make it difficult to determine which parts of the data model are still in use, and to make sure that all annotations were captured.

[58] Readers with software experience might note that the format of this physical model is an ER Diagram. In another work, I link the ER diagram to the maintenance of whiteness (Stevens et al., 2021)
[59] Code becomes "legacy code" when the functionality it offers is no longer used by the application. This code is often not deleted for several reasons, including that developers are unsure if deleting it would break other functionality.

Figure 21

The Lovense physical database.



Second, some table names do not make sense in the scope of the application. For example, a table named BackWork does not seem to connect to anything. In this case, a trip to the file where that table was defined is not any help. The file BackWork.java simply contains additional getters and setters.[60] Lovense uses the common data access object (DAO) pattern, that allows developers to separate the functionality of a data object (in this case, a table) from the fields and field storage information. In the case of mysterious tables, visiting every file in which their parent classes are mentioned is the only thing that I can do. BackWorkDao.java does not contain any additional helpful information, unfortunately. It is not uncommon for production databases to contain legacy tables and for production codebases to contain legacy code. That could be the case here. However, because the decompiling process is imperfect, it is simply not possible to be certain.

Third, some tables, like tb_program_pattern do not seem to have relationships with other tables. Some tables have a primary key that seems like it is keyed based on a text field, often called "owner", rather than the more common id field. In this case, I speculate that it is keyed off the userid, based on a field named "ownerEmail" in table tb_alarm. Some field names also do not make immediate sense, and in some cases, deeper understanding is complicated by the initial obfuscation of the code---function and variable names are turned into long strings, obscuring their purpose, making code especially difficult to read. Given these difficulties, further examination of the code is necessary in order to move to a conceptual data model.

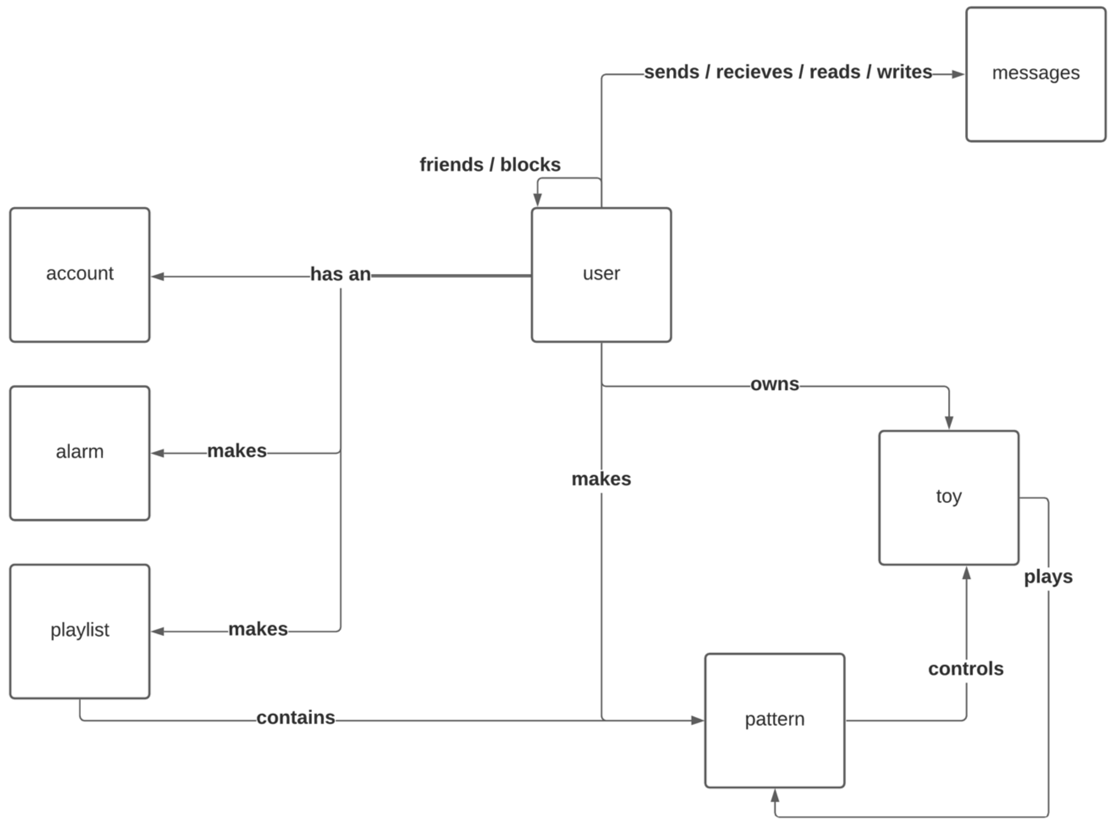
[60] In object-oriented programming, getters and setters are methods that get (return) or set (establish) a particular property of a class.

Making a conceptual model

The final step in reverse engineering Lovense Connect is to develop a conceptual model. The diagram in Figure 22 is a conceptual data model---representing the entities of concern in the system. There are many ways to create a conceptual diagram and Figure 22 represents one way to explain what is happening at a high level (aka distinct from the database's physical model depicted in the previous step).

Figure 22

A conceptual model of the Lovense application.



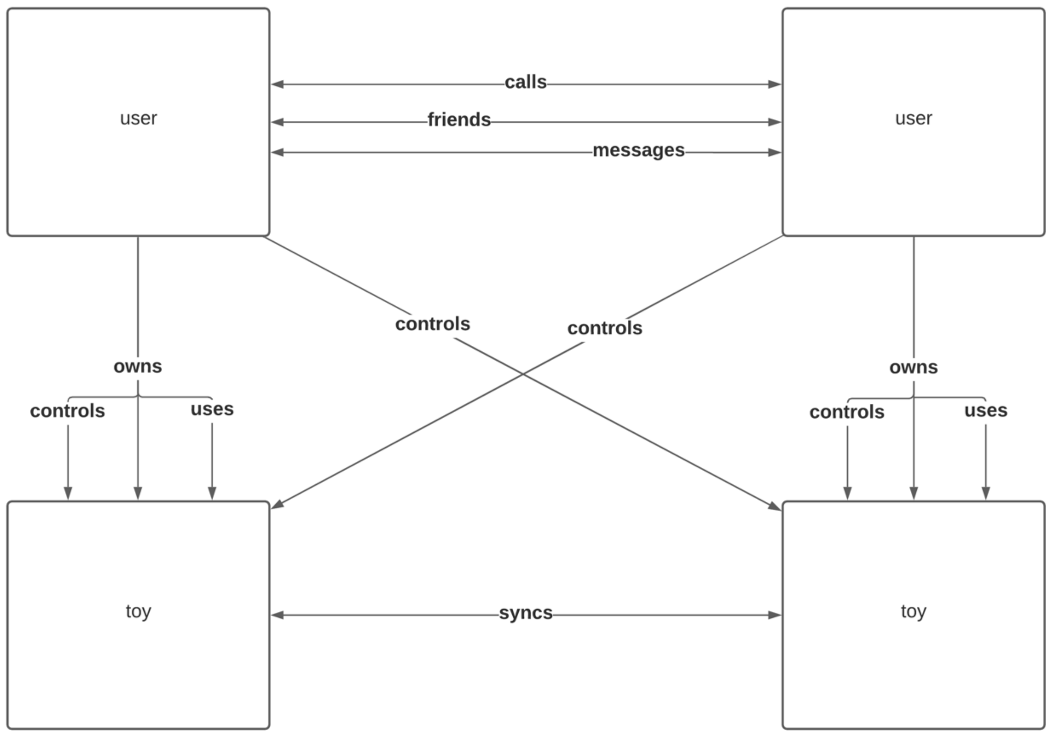
In order to understand a conceptual model, it is important to understand the work that the shapes and lines are doing here. In Figure , the squares simultaneously represent abstract entities and all instances of those entities. That is, the <u>user</u> box is both an entity describing every human user, and a representation of how one particular user might be connected to other parts of the application. The lines represent one expression of the relationships between entities. Parts of Figure could be read aloud as "Users own toys" to signify that user entities have an owner/owned relationship with toy entities. It could also be read as "A user owns toys" to signify that the relationship is not necessarily a 1:1 relationship. It could also be read as "Users friend and block other users" to indicate that individual users can perform actions (becoming friends with, blocking) other users.

As these read-aloud examples indicate, the words on the lines are descriptions of the relationships between entities. The relationships are bi-directional, e.g., users own toys and toys are owned by users. However, the relationship is depicted in the direction that favors using active-voice verbs. Within an object-oriented paradigm, the active voice signals to the programmers which entities might "own" the tasks related to maintaining the relationship. A conceptual diagram might be considered complete when reading it aloud covers the main functionality of a product. In the case of the diagram above, there is more detail missing about the ways that users and toys can interact with each other.

Figure 23 helps to fill the user-toy gap. Here again, squares represent entities and the lines represent some of the relationships possible between those entities.   In this case, the polysemous line between user and toy in Figure 22 has been expanded to show how two users, each with their own toys, can interact. This diagram can be read just like the previous diagram -- "a user messages a user", "a user controls a toy." However, this diagram is itself also an abstraction leaving out the technical details and other entities that might be involved in relationship/actions like "message."

Figure 23

A conceptual model of Lovense user interaction.



While the steps I have outlined may seem especially technical or tedious, within this project, they are essential to developing a deeper understanding of the software and the context in which it is used. Especially for researchers without experience creating commercial software, having a first-hand experience of the multitudinous transformations and mediations involved is crucial. With this information, we have immensely more information than we had without it.

Conclusion

The crucial takeaway from this reverse engineering processes is that software and its internal operations do not need to be taken at face value. Once data enters a database, it often enters a context of normalization. Normalization is both decontextualization (via the separation of parts of data into different tables) and a generalization (via the elimination of field dependencies. However, the power of decontextualization is not only in the specific harms that it causes to those using the software, but that through the use of decontextualization, certain things stay frictionless, seem as though they are the only approach, and support how it has always been done. In the data model, it is possible that there is not oppression, but smoothness, ease and the infrastructural invisibility of white-cisness. As a result, looking through these data models is the opposite of reverse engineering to discover bias. Instead, the reverse engineering helps us to discover ways that things have always been, ways that things stay smooth. In some ways, this is reverse engineering to see how "well" something is made. This is a critical perspectival shift.

By looking at the data models for Lovense, we can see the normalization of surveillance and the generalization of everyone who uses the product (whether they be cam workers or lovers) into one entity of Users. We can also see the exceptional lack of security and attention to privacy in the data model. Though there maybe be some small security allowances in the code itself, those allowances are absent in the model, which is, I believe, a reflection of the social location of the modeling team. In this chapter, I demonstrated a methodology for recreating the physical data model of one piece of commercial software, the Lovense Android application. This methodology is an important contribution to the fields of science and technology studies, critical data and critical software studies.

CHAPTER 4

USING MODELS AS INTERVENTION

Passwords must be at least 8 characters.

Password must contain at least one capital letter and one number.

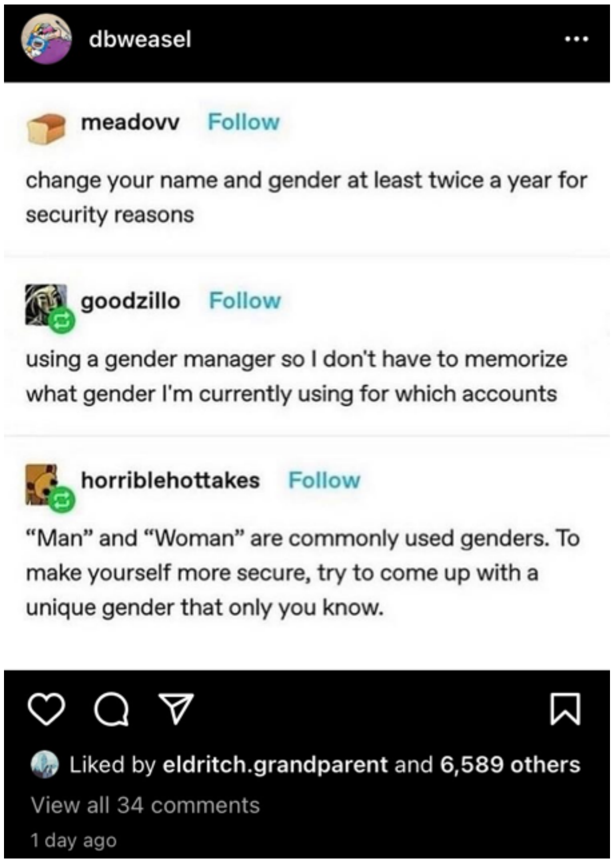
Password must not be a dictionary word.

Software systems that require you to login will often present you with messages about password requirements when you register. Those requirements are designed to encourage you to choose a "strong password" (aka, one with many different characters in it or one that does not contain your username). In contrast, a "weak password" is one that is easy to guess or trivial for a simple computer program to hack. We need strong passwords to protect our access to our accounts and to keep our information secure. The increased awareness of password security requirements, and the general development of security culture informed a quote posted by tumblr user "meadovv." They wrote: "Change your name and gender at least twice a year for security reasons" (See Figure 24). Meadovv's comment was the first in a series of comments riffing on software security requirements. But meaddovv and the other users commenting are not simply mocking password requirements, they are demonstrating a sophisticated understanding of gender through a trans perspective, satirizing the limits of technology's capacity for handling trans genders.[61]

[61] I am not making any claims as to the gender identities of the people behind these unknown-to-me tumblr accounts.

Figure 24

An Instagram post from user dbweasel of a tumblr meme.



dbweasel

meadovv Follow

change your name and gender at least twice a year for security reasons

goodzillo Follow

using a gender manager so I don't have to memorize what gender I'm currently using for which accounts

horriblehottakes Follow

"Man" and "Woman" are commonly used genders. To make yourself more secure, try to come up with a unique gender that only you know.

Liked by eldritch.grandparent and 6,589 others

View all 34 comments

1 day ago

In this chapter, I explore technology's limits for handling genders. I begin with data models, demonstrating how data models can be used to intervene in the way software handles aspects of people's lives like gender. I then discuss the limitations of this approach, acknowledging that the involvement of technology does not *improve* marginalized peoples' experiences with datafied systems, only makes those experiences less bad.

To return to the tumblr post, meaddov's joke that we should change our name and gender twice a year for security purposes reveals an understanding of gender as fluid, and perhaps even capricious and hard to pin down. Tumblr user "goodzillo" riffs on meadovv's comment: "using a gender manager so I don't have to memorize what gender I'm currently using for which accounts" (Figure 24). Goodzillo's comment gestures toward the password managers so often recommended as a help to tracking complicated password requirements. Passwords are contextual: we should not use the same password for different accounts. If one of those accounts gets hacked, your password could be guessed and then used on another site. In order to keep track of all of these different passwords, it is helpful to use a password manager to keep track and reduce the temptation to reuse easy-to-remember passwords. Here goodzillo is joking that they have so many genders (one for each account) that they need to use a gender manager to keep track. Their comment acknowledges the contextuality of gender, and pushes meadovv's offering, encouraging us to think more deeply about gender and technology together, and offers space from demands that trans people have one consistent and known gender. The subsequent comment from user "horriblehottakes" reinforces this when they say "'Man' and 'Woman' are commonly used genders. To make yourself more secure, try to come up with a unique gender that only you know."

Trans people often must perform "wrong body narratives" in order to access gender affirming care, making the possibility of changing one's gender twice a year especially absurd. In this context, as Jay Prosser notes, transsexuality is "always narrative work" (Prosser, 1998, p. 4). These narratives (which absolutely may feel authentic to some) create a trans normativity that conceptualizes transness as problem with a solution---surgico-hormonal intervention, then, is the a way of making a wrong body into a right body (Hughes, 2018). Within this narrative, the bodily corrections happens only once and these ideas of transness are founded on and inseparable from teleology: transness is a directional and goal-oriented movement from one gender to another. However, this teleological foundation of transness does not stand alone. As Chen and cárdenas note "If trans implies a movement from one gender toward a different location, then transness is always imbricated with forward time and cannot exist without linear, teleological time" (J. N. Chen & cárdenas, 2019). Linear, teleological time is what Jacob Lau describes as "cis normative time." Drawing on the recognition that transness is racialized, we can say then that this is not just "cis normative time," but "white-cis normative time." Thus white-cis time "remains the master code through which trans folks must translate their racialized, gendered and sexed histories" (Lau, 2016, p. 2). This translation process is founded on ideas of the fixity of gender, and its inherent knowability.

White-cis time also manifests in infrastructural control over trans life. A major part of many trans people's lives is working to minimize the friction happens when a trans person encounters a datacentric and white-cis centric infrastructure. For example, as trans people were able to access gender-affirming care and to live in alignment with their gender, they needed documents to reflect their gender.[62] Accurate documentation is, among many things, an access issue--a trans woman with an id reading "M" can be subjected to "embarrassing and invasive questions" when trying to simply go about the business of her life (Davis, 2017). Through the efforts of trans activists, the amount of paperwork and gate keeping to change one's gender marker on state documents has steadily decreased in most states. However, for many of us, neither an M nor an F is the correct gender marker. Beginning with Oregon in 2017, people like me were able to access X gender markers on driver licenses; and in 2021, the US Government began issuing passports with an X (Hauser, 2021; Wamsley, 2017). Whether or not there should be gender markers on state documents is a matter well-covered by other scholars (Davis, 2017). What is important here is that gender

[62] For an overview of the administration of gender see Spade (2008). Cassius Adair (2019) traces the development of driver licenses and administrative gender changes through the lens of anti-Blackness in the US.

markers reflect the state's desire to administer, supervise and receive "proof" of an individuals's gender marker change. In other words, once an individual has been proven an "authentic trans," the state administers gender (transness) using the logics of white-cis time. This administration imagines transness as a simple state-change-over-time: a document sex marker is changed from F to M or from M to X and it assumes that once someone has become an M -- legally, medically, socially -- no other change is expected and they are to remain an M for the duration of their lives; no gender manager is required.

While this undated exchange between three tumblr accounts is satirical, it is also illuminating. This exchange demonstrates a sophisticated understanding of gender. In my reading, the idea of changing one's name and gender twice a year is inherently trans. However, transness is not simply about a change in gender and (for some) name.

To be trans in the United States is to be always already ensconced in the racialized and gendered logics of colonialism. White supremacy and colonialism created ideas of cisgenderedness, and so to call yourself trans (as I do) is to be simultaneously participant in and victim of white supremacist and colonialist logics. In the United States, conceptions of transness are inextricable from race.[63] As Emily Skidmore recounts trans woman Christene Jorgensen's rise to fame in the 1950s, she describes Jorgensen's embodiment of whiteness, heterosexuality, and conventional femininity as inextricable from her acceptance in popular media of the time. In other words, Jorgensen only counted as a "good transsexual" because she conformed to normative expectations of white womanhood. Other contemporary trans women---Mexico City native Marta Olmos Ramiro, Pacific Islander Laverne Peterson, and Black American Delisa Newton---were exoticized and rendered deviant, described mockingly

[63] A history of trans in any context is out of scope here. For more trans histories, see Meyerowitz (2002), Skidmore (2011), Snorton (2017),Stone (1992), Stryker & Aizura (2013).

in the press. In an autobiographical essay, Newton tells readers how she struggled to find gender affirming care: “Because I am a Negro it took me twice as long to get my sex change operation as it would have a white person. Because I am a Negro many doctors showed me little sympathy and understanding” (Skidmore, 2011, p. 292). Newton's statement, then, provides one small example of the way that transness, both then and now, is always racialized.

The comment thread by meaddov, goodzilla and horriblehottakes gestures toward a speculative ecosystem, in which gender, and perhaps sexuality and queerness, can exist comfortably within sociotechnical systems. In “QueerOS” Fiona Barnett, et al. build on Kara Keeling's work to think queerness and digital technologies together. “For Keeling, a QueerOS would make it impossible to think of phenomena of identitarian difference as separate from information technologies” (Barnett et al., n.d., p. 1) What would it mean to have a queer operating system, a QueerOS, that conceives of queerness as socially constructed with aspects that mirror computational operating systems, like interfaces, users and kernels. The intervention in QueerOS is to offer new perspectives of queerness, but also, and crucially for my work, to address “the challenges of imbricating queer/trans/racialized lives and building digital/technical architectures that do not replicate existing systems of oppression” (Barnett et al., n.d., p. 2) However, a QueerOS is only “theoretical vaporware, speculative potentialware, ephemeral praxis” (Barnett et al., n.d., p. 2). In this chapter, I take up the challenge of thinking racialized trans lives, not through vaporware but through a built data/software ecosystem.

If we think of transness, and the multiplicity of trans lives to be less teleological and instead “about a multidirectional movement in an open field of possibility, then time and its direction become more fluid.” (J. N. Chen, 2019, p. 473). Computing and its objects, including databases and data models, do not lend themselves to fluidity. However, in this chapter, I test the limits of data models to build a collection of entities that are fluid, evolving and perhaps, fundamentally unknowable. This critical engineering intervention will serves as a way to both imagine humane technology and as a "return the possibility of imagining alternative futures" (Parikka, 2013, p. 51).

Modeling (trans)gender

"Instead of speaking of a person's gender status, we might do well to speak of a person's gendered status in a given cultural location, at a given time, and for a given purpose" (Hale, 1997, p. 232).

Because the temporality of my own work and critical technical practice is oriented towards real situations that need intervention, I begin this section with a vignette. This vignette is a composite of many people's experiences --- my own, those of my friends and of people in my communities who have shared their struggles when gender encounters computational systems. This vignette also mirrors the practice of user stories and user experience personas from previous chapters.

———

On Monday, Tawny went into their local clinic for an HIV test. They gave their name and contact information, and River, a clinic employee, entered that information into their client database. After the test, Tawny and River talked about how the clinic sometimes needs volunteers to help hand out condoms and HIV literature. Tawny was interested so River put their same information into their volunteer management database.

On Friday, River called Tawny at home to invite them to volunteer because they had an urgent need for help that night.

"Hi, is Tawny there?"

"You have the wrong number."

"Oh, no I don't think I do. They're really tall, with brown hair and a heart tattoo on their forearm."

"Ohh, that's Marcus. He'll be home in an hour."

The following night, Tawny is performing as a drag queen. River was out with his friends and recognized them making the rounds of the tall cocktail tables in the bar.

"Hi Tawny!"

Tawny shot him a look, "When I'm here and when I'm dressed, you better call me Miss Anthropy."[64]

The next morning, Tawny wears their hair down, puts on a dress and goes to brunch with friends. On the way home, they stop at the pharmacy to pick up their prescriptions. The pharmacist tells them that only Marcus can pick up these controlled substances. Tawny shows the pharmacist their (Marcus's) driver license. The pharmacist spends a long time looking at Tawny, and looking at the ID, finally agreeing to give them the prescription. Tawny is relieved until they hand the pharmacist their TrueName MasterCard to pay.[65] The pharmacist, seeing the name difference, refuses to give them their prescriptions.

———

In the current way of storing gender (as a binary attribute of a Person entity), there is no way to capture Tawny's experience or to ask questions about their gender that give us meaningful and useful (never mind, true) answers. What Tawny's story illustrates for us is

[64] Thanks to Mikel Wadewitz for this drag name

[65] TrueName is an initiative by MasterCard allowing trans people to have their "true" (as opposed to legal name) on their debit and credit cards.

just how high the stakes are for understanding gender as contextual and temporal---for pushing past white-cis transition narratives and for having those understandings reflected in our administrative/database systems. Places where non-cis people meet white-cis data structures are sites of administrative violence and data violence (Hoffmann, 2018; Spade, 2011). The modeling that I do next, then, is not a recommendation for the datafication of everyone's gender as a self-evident good end goal. Instead, it is an immediate practice-based approach to harm reduction for people like the fictionalized Tawny and the very real trans people like me and my community. In what follows, I demonstrate step-by-step the epistemological and data shifts from Figure where gender is an attribute to its final form where gender is regarded as temporal, contextual and multiplicitous, stored in a way that actively subverts white-cis linear time. This demonstration is also important because, for those who have never made a data model, it shows how many active and conscious choices need to be made in its creation. Moving stepwise through this process reinforces the data model as transformation instead of abstraction and demonstrates the crucial move of promoting attributes to entities. Furthermore, it demonstrates that data models are important sites of intervention. As I demonstrate in this section, changing the data gender model also changes the shape of the software that uses that model.

Making the Model

Drawing on the legacy of trans narrative as method, and the grounded nature of critical technical practice, I begin this part using a statement (or a proto user story, to use the terms from Chapter 1) from the opening vignette: *Out and about, Tawny uses they/them pronouns and their gender is woman.* Initially, this statement could be represented by two attributes on a user entity (see Figure 25). However, this is the same representation that has been failing us and it does not account for the complexity of Tawny's life. Contrast this with Figure 26. Similar to the address model, we now have support for a Tawny to have as many genders as needed. This lets Tawny have genders like woman, man, drag queen, and femme. In this simple act—promoting gender to an entity—we are able to open space for gender to become (within relational onto-epistemologies) an object distinct from a person, and with its own attributes and relations. Although we are still working within the limiting factors of onto-epistemological commitments, by objectifying ender, we separate it from the person and use the power of relations to position gender as a subject with its own set of attributes.[66] This is an example of the promotion of an attribute to an entity that I discussed in Chapter 2.

[66] I am excited about future interpretations of this move through the lens of abjection as framed by Robert Phillips in Stryker & Currah (2014).

Figure 25

A one-table model showing the attributes of the User entity.

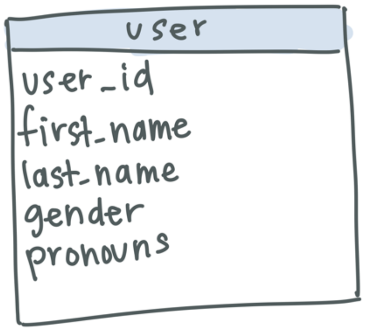
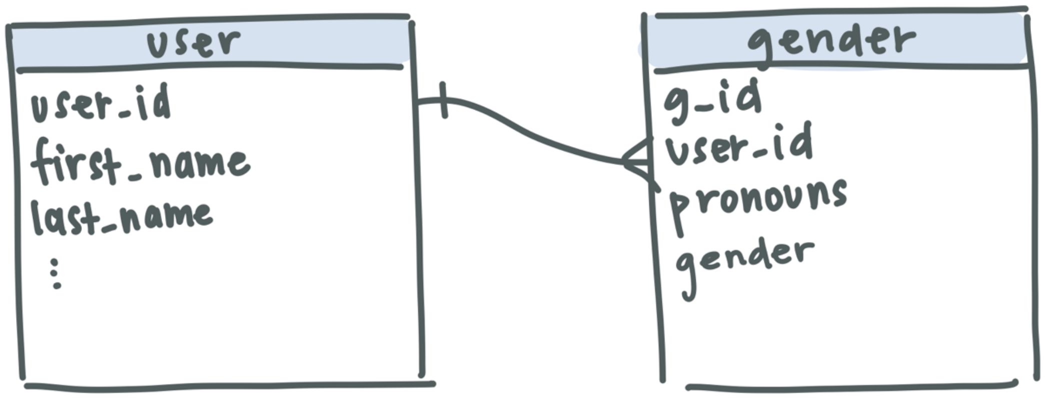


Figure 26

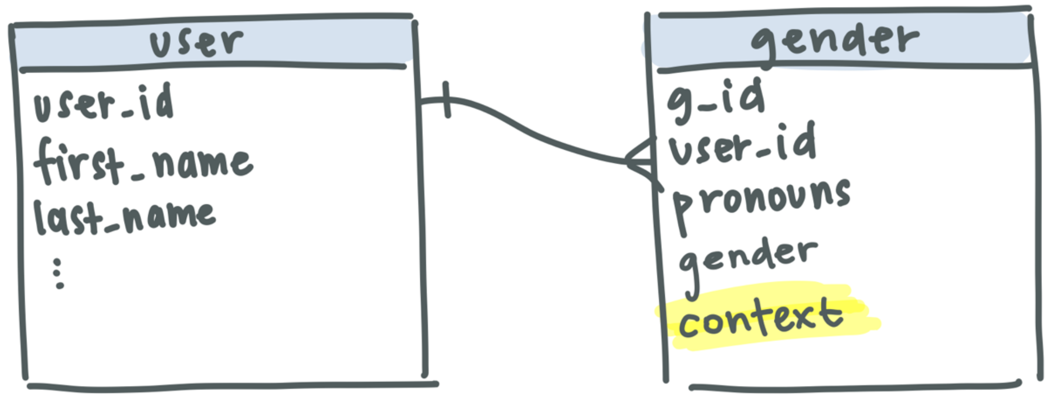
A two-table model, promoting the Gender attribute to an entity.



However, the diagram in Figure 26 does not support the nuance of Tawny's gender, which we know to be fluid and contextual. Figure 26 supports us asking important questions like "What genders does [person] have and what pronouns do they use for each?" But it lacks the flexibility to tell us more than that. As Tawny's story shows us, gender is not just about having more than one set of pronouns, it is about understanding the contextual nature of that set. How can we record information about the context in which a gender is active? As a next step, we can add contextual information to the gender table by making context an attribute of gender (see Figure 27). Adding context as an attribute of gender is a logical next step -- it needs to go somewhere and since it's about gender, this seems fitting. However, this design choice also implies that context is gender dependent, rather than gender being context dependent. This is a subtle but important difference related to what becomes an attribute. When context is relegated to an attribute of gender, it can imply that context does not exist separate from gender and that two genders might not have the same context, or two contexts the same gender. This is the same logic undergirding the move to make address its own entity -- it exists separate from the user.

Figure 27

A two-table model, with context as an attribute to the Gender entity.

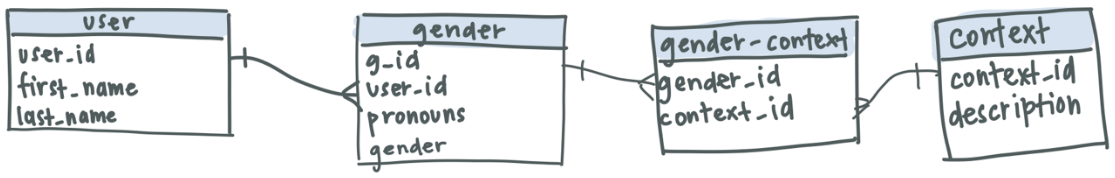


This choice assumes that a person can have an unlimited number of genders, but that each gender exists within its own context and has its own set of pronouns. The design in Figure 27 replicates the flattening that we are already working to subvert--- the design assumes one set of pronouns in one context. That is, it requires that when I am at a particular place and my gender is nonbinary, I use they/them pronouns. If there are several contexts in which I have this gender/pronoun combination, the design would create redundancy and violate the rules of normalization.

Instead, consider Figure 28. Here again, we have made the crucial onto-epistemological move of promoting an attribute (the context field) to an entity (the Context table). As we have seen in the past, this promotion creates opportunities for considering what kind of information about a given context would be useful to store. Figure 28 is a minimalistic example, adding only the field "description." I could imagine a Context table that contains many more fields about a particular context, as appropriate to the software's context of use. Note that Figure 28 has two new tables, Context and Gender-Context. This second table is necessary because it links one context to many genders, and one gender to many contexts. In other words, this table allows for a gender to exist in more than one context, and in turn, for a context to contain more than one gender.

Figure 28

A four-table model, in which Context has been promoted to an entity.



This move reflects the position that the pronouns one uses do not need to align with the normative expectations of that gender. To keep pronoun as an attribute of the gender entity reinforces this connection in cisnormative ways. The separation of pronouns into a linked table, reinforces the contextual nature of pronoun usage, and this series of design choices-based on the intentional placing of attributes, opens the possibility of allowing the data model to let gender be as fluid and contextual as we known it to be, and to consider "gender status" rather than a fixed gender. This diagram also does not make an explicit allowance for multiple sets of pronouns. Are he/they pronouns the same as having two separate entries, e.g., he/him and they/them entries? It is not for me to decide whether people experience difference between describing themselves as "he/him or they/them" or "he/they." This design could support both, depending, as always, on the questions that the database needs to answer.

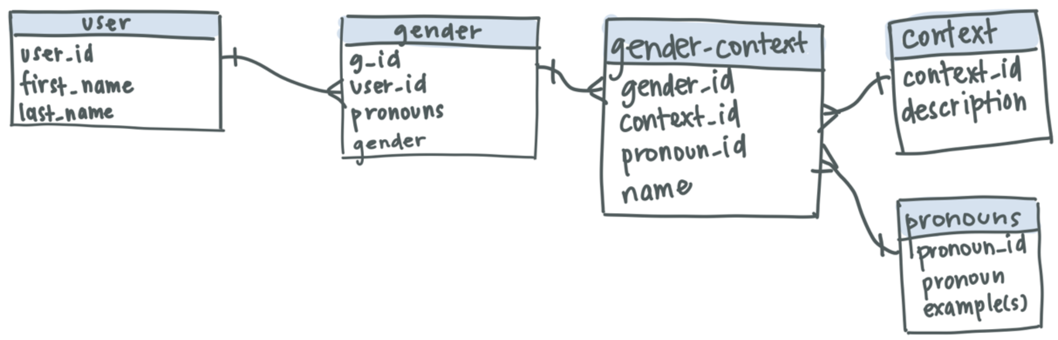
However, the model is still not complete. Currently <u>gender</u> and <u>pronouns</u> are sibling attributes on a gender entity. However, this assumes that a gender always uses the same pronouns, even in different contexts: this assumes that in every place where my gender is trans, I use they/them pronouns. However, this is not the case. In order to store this in our current configuration (that is, in the model represented Figure), I would need to have two different genders, one with they/them pronouns and one with he/him pronouns. This assumes that the genders are ontologically different when using different pronouns. The appropriate next move, then, is to turn pronouns into their own entity as well. Because there are a finite number of pronouns possible, it makes sense for pronouns to be a simple lookup table, listing the pronouns, and even an example sentence or pronunciation (for neo-pronouns).[67]

This chapter opened with an anecdote about changing name and gender twice a year for security reasons. Name is connected to gender and to context, as the vignette with Tawny illustrates. We need a way to associate name with gender, context, and pronouns. Figure 29 does exactly this by adding the name to the <u>Gender-Context</u> table which now contains gender, context, pronouns, name and (by extension) the user record itself.

[67] I want to note that this is an arbitrary design choice. I prefer a look up table when the number of options is likely finite and knowable in advance (even if we consider adding pronouns in other languages), but there is a valid case for designing this another way.

Figure 29

A five-table model, adding name to the Gender-Context entity.



This final iteration allows us to represent gender statuses from the vignette: When Tawny (user) is at the club (context) her name is Miss Anthropy (name), they are a drag queen (gender) who uses she/her pronouns (pronouns).

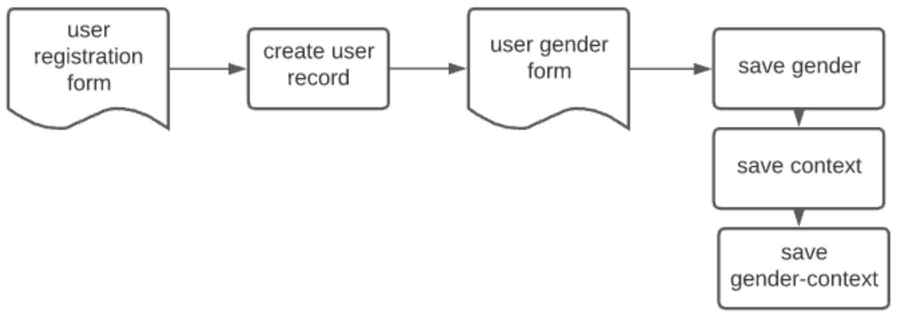
While this model is clearly abbreviated to show only the attributes of interest to our discussion, in my design this data model does not contain timestamp fields. The presence of timestamp fields indicates that the order in which data were entered is important, and that fields like date_created provide useful information. Those fields are often proxies for "records that are new" or to answer questions like "Is this user's data stale?" In this model, I advocate for other ways of answering these questions. For database systems that automatically add timestamps, I recommend deliberately obfuscating and scrambling that data both on insert and on update.

Implementing The Model

The model that I made in the previous section (Figure 29) is built using basic relational paradigms, and it is able to fit in to existing relational databases. However, since models do not exist in a vacuum, it is useful to see how it could be implemented to collect the data it has been designed to store. Because this model has four new tables of user data, the order in which data is saved becomes important. Because gender is a separate entity from user, the user record will need to exist first. This workflow might look something like Figure 30. In other words, the promotion of gender to an entity allows it to be completely separated from the creation of a user record, almost as if the user's gender was not central to their existence or database validity.

Figure 30

An order of operations for entering data into the full data model.

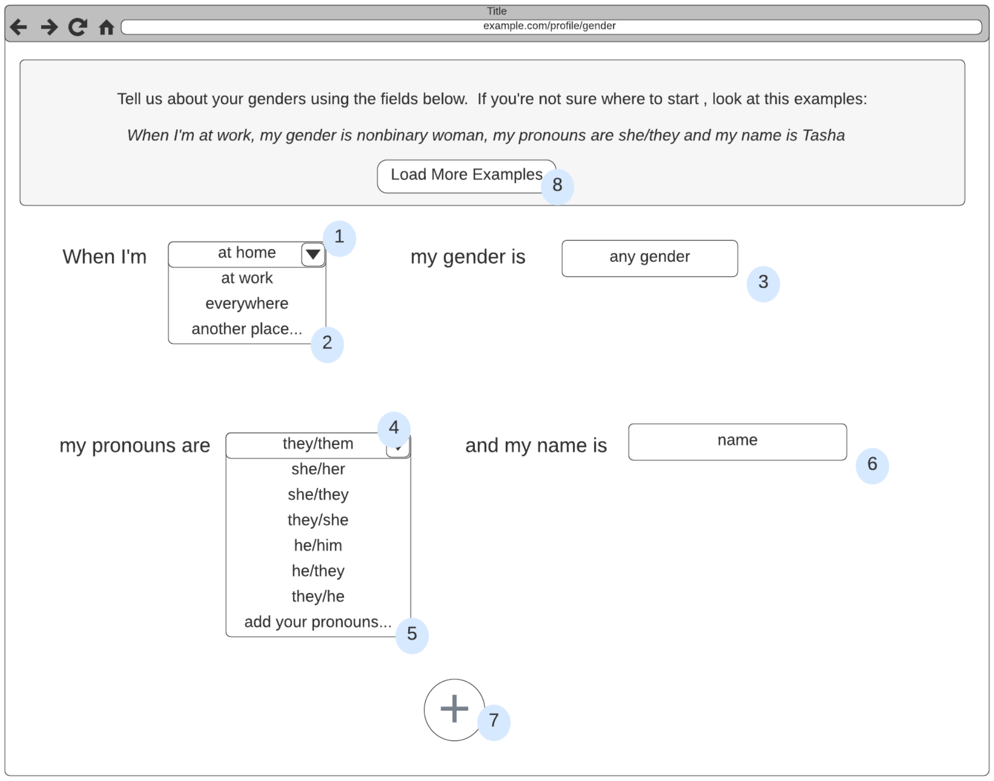


In order to visualize how different this data model is and to continue to reinforce the ways that data models shape software affordances, it is helpful to look at a sample implementation. In software implementing this data model, gender collection might look like the wireframe in Figure 31. A wireframe is "the basic blueprint that illustrates the core form and function found on a single screen" of an application (Hamm, 2014, p. 15). Because of the data model pictured in Figure 29, the software contains affordances that allow the user to represent their gender(s) in a way that is quite .similar to how they might describe them during a casual greeting, or aloud to a friend: "I'm not out at work, so I use he/him pronouns and my coworkers call me Trevin." This form could appear in an "edit profile" section of a website, as part of a user onboarding process, or other appropriate context. The wireframe itself contains only minimal user interface elements as a proof-of-concept and a visual contrast to some of the more typical ways that gender is requested in software products. Throughout this dissertation, I have argued that data models have immense power in the ways that software is shaped and built. This example supports that argument: by changing the data model of gender, the software's shape will change accordingly.

There is an important question here that I leave deliberately ambiguous both in the data model and in the wireframe --- while some people have different genders in different moments or different contexts, others experience significant gender continuity (or, perhaps the opposite of gender fluidity), but use a different name or other pronouns. The presence of the gender field doesn't require people to have distinct genders when other factors change. This is the point of this design choice. It allows for a separation between name, gender and pronoun that supports both people who have different genders and people who, for infinite reasons, use different pronouns or names (and similarly, who might use the same name and pronoun, but have different genders).

Figure 31

A wireframe showing a proof-of-concept implementation for the gender model



Tell us about your genders using the fields below. If you're not sure where to start , look at this examples:

When I'm at work, my gender is nonbinary woman, my pronouns are she/they and my name is Tasha

This wireframe starts with example sentences demonstrating to the user that the software supports a rendering of themselves that is similar to how they might want to describe it. The button (**8**) allows the user to load more example sentences. The form replicates the structure of the example sentence above, encouraging the user to follow a familiar "mad libs" style fill-in-the-blank. This implementation invites users to firstly consider the context (**1**) when answering questions about their gender. The context field is a dropdown pre-populated with several (included in the wireframe as example) common contexts as a start for the user. The user is not required to choose one of these contexts, but can choose "another place" and enter a context that is appropriate for them. The field is called "context" and not place because context is a more expansive term that may include phrases like "anytime I'm with my family" or "when I'm in drag" - ideas that transcend geographic notions of place. Next, the user can enter their gender in a text entry field (**3**). Advocates of "clean data" prefer set options to text entry because people could enter "anything at all" into these fields. From my perspective, being able to enter "anything at all" is exactly the point; clean data is as oxymoronic as raw data (Gitelman, 2013). The pronoun field (**4**) mirrors the structure of the context field---users are presented with several common answers, but users are free to enter their own answer. In this configuration, those answers do not become part of the list available to other users - it is not a crowd-sourced list of pronouns. This is in part because for some people, their pronoun is their name, and they decline any third-person pronoun usage at all. The name field (**6**) exists within the form asking about gender. This could imply that the user's name is not requested in other parts of the onboarding (but again, this is a proof of concept.) Finally, the form invites users to add another statement (**7**) as needed. The wireframe is missing the mechanisms by which users can edit or delete pre-existing records/entries, though that functionality is obviously necessary for production software.

In this section I have presented a data model in which aspects of a person's gender(s) are taken separately and seriously, resulting in a model that, on its face, does not reproduce some of the harms that sociotechnical systems can cause trans people. The wireframe that extends from the model supports this -- the data model results in a gender entry form that can start to capture the contextuality and fluidity of gender. But now that it is possible and complete, is it actually a thing that should get built? Is the answer to reducing systemic, infrastructural and data harms to give *more* data? To be more visible? Frameworks of harm reduction are helpful here.

Analysis

Recall the story of Tawny and their interactions at the health clinic, at their house, at the drag bar, and at the pharmacy. In one case, the clinic's database's flat approach to gender lead to Tawny being misgendered both at home and in public. In the case of the pharmacist, Tawny was not able to receive the medications that they needed. One database would not solve both problems, of course; but if both the clinic and the pharmacy had implemented versions of the data model and wireframe I described (in Figure 29 and Figure 31), Tawny may have had a far better experience.

However, in this chapter, I have taken that demand seriously and made a working data model and functional wireframe that attends to the real harm that trans people experience during their encounters with datafication, identification and categorization.

My offering does reduce some of those harms and is thus best considered within a framework of harm reduction. Harm reduction is often described as a method of "meeting people where they're at," typically in the context of drug use and further harm or disease prevention. In that context, harm reduction is based on the belief that drug use harms people and it works to mitigate that harm, rather than encourage people to stop using drugs. Harm reduction programs provide interventions that often include needle exchanges and safe drug use sites. In my work, I think of harm reduction as a reformist approach, acknowledging that "where society is at" is a profoundly oppressive and dysfunctional moment for most groups of people. Rather than move to abolish those oppressions, the approach in this chapter seeks to mitigate the harm caused by whiteness and cisness.

In "Decolonization is Not a Metaphor," Eve Tuck and Wayne K. Yang describe the process of *settler harm reduction* as one "crucial in the resuscitation of practices and intellectual life outside of settler ontologies" (Tuck & Yang, 2012, p. 21). In that lineage, this chapter is a form of white-cis harm reduction, rehydrating ideas of gender in data-centric conversations and circles. However, I heed their caution that forms of harm reduction, regardless of how important they may be "[do] not inherently offer any pathways" to fundamental structural social change.[68] Harm reduction does offer material changes to the present world, but those changes, like this data model and wireframe, are contextual just like the genders they are capturing.

There are four specific harm reduction benefits. First, the data model creates affordances separating the user from gender data. Via this separation, the user table could be queried and user data, absent gender, would be returned. This turns the user's gender into a sort of NULL value---unknowable until other tables are queried. The NULLness of gender here renders it unknowable, indeterminate and, in that way, it could be considered queer (Gaboury, 2018)[69]. While this may seem to be simply a theoretical benefit, a programmatic/data-based separation between sets of values cascades into other aspects of the software.

Second, because of the way that the model leverages Codd's relational paradigm, it offers capacious affordances for the ways that a user might experience their gender(s). Relational moves (or, even the paradigm of normalization) are often used to advance logics of scalability and extensibility. While I am neither interested in nor advocating ideas of "scalable gender," this model allows for the infinite ways that individuals experience their contextual gender(s). With this allowance, the model responds to critiques of datafied

[68] One method of technosocial harm reduction is the series of initiatives designed to store Indigenous place names for locations in the United States. These toponymic projects often use USGS GIS data as the foundation, and then overlay Indigenous words for places (Chambers et al., 2004).
[69] My thinking here is also influenced by Bopp et al., (2019), Johnson (2018).

systems in which people are forced to commit themselves to a gender which is not something they experience full time.

Third, the model especially rejects a popular technical move toward conciseness and brevity. Computing and data modeling best practice is traditionally to store information in the smallest container possible, abbreviating and curating wherever appropriate---states and countries are often stored by abbreviation and then hydrated for display to the user; formatted numbers, like phone numbers and social security numbers, are stored only with numbers and formatted for familiar display. By promoting so many aspects of gender to the status of entities, the model reflects an epistemology that there is an expansive amount of information that is relevant to gender. It signals that brevity is a disservice to the subject and it follows the admonition to be careful "that we have not lost any data in the translation to tabular form" (Simsion & Witt, 2005, p. 41). In other words, via its design, it acknowledges that so much data has been lost and is a move to restore some of it. Within a datafied context, this expansive epistemology implies that promoting other aspects of one's identity (race, disability) to entities also invites the restoration of lost data.

Finally, this model grapples directly with white-cis temporality and its limitations and affordances. While data models are designed to support answering questions, their design can also resist providing answers. Because there are no timestamps or integer-based primary keys, the model resists answering questions about transition "direction," treating all entered genders as co-emergent.[70] One of the motivations of including gender on identification documents was to prevent "fraud" and to make the identified person "known" to the identifier (Currah & Moore, 2009). That "fraud" was a concern reflects the belief in a "truth"

[70] A common question that cis people ask when encountering a person who just says that they are "trans" is to ask "from what to what?" expecting an answer like "FtM" or "MtF"

that that the fraud might hide. This data model promotes multiplicity, fluidity and a structural acknowledgement that many authentic statements can exist simultaneously.

This data model was borne out of my deep concern for the harms that sociotechnical systems cause marginalized people. However, harm reduction approaches do not reduce harm equally. Aligned with statist biopolitical interventions, the data model separates gender from other aspects of race, class, disability, and sexual orientation. For many cultures, one's gender identity and sexual orientation are inseparable and one word would be used to describe both. Similarly, certain words for gender identity are available only within certain racial and cultural milieus. The designs of the form and model do not reflect those imbrications. This is an example of one of the myriad ways that the model exists within white, western ideas of fixity and knowability, and is fundamentally an assimilationist approach, performing the data version of respectability politics. In this way, it is reminiscent of the ways that the trans people, like Christine Jorgensen, who rose to early notoriety were the most cis- and hetero-normative people. It is for women like Christene Jorgensen, and subsequent decades of white, cishetero normative trans people that shifts in the administration of gender have happened.

Writing about X gender markers on passports, Christine Quinan asks provocatively "Could we see these changes as the individual pushing back against the nation state, pushing the boundaries of the state, and the state responding and acknowledging self-expression?" (Quinan, 2017, p. 164). In other words, are more nuanced gender identification and administration systems good? Do they give us more power when interacting with the state? I argue that the answer is an unequivocal "No." As Eric A. Stanley has argued, we should "resist reading these biopolitical shifts as victories" and resist any "work to translate and in turn confine the excesses of gendered life into managed categories at the very moment of radical possibility" (Stanley, 2014, p. 90). The data model in this chapter does just that---it enacts, within the limits of available technology, one understanding of how gender works and incorporates many of the critiques that trans people have about gender collection systems; it turns our gendered lives, in all of their messy excesses into clearly defined attributes of entities.

Implementing this data model is a profound act of settling, an act in which "*less bad* becomes the only name freedom can take" (Stanley, 2021, p. 115 emphasis in original). That this data model is *less bad* does not make it *not* bad and does not make it better than inviting moments of radical possibility. In the absence of data models that "accurately" capture a person's gender and its expansive constellation of facts, we are free to dream both abolitionist futures and alternative interventions. Rather than doing a better job of collecting gender, we can, in nearly all contexts, simply do no job at all (Cifor et al., n.d.).

CONCLUSION

In October 2018, Stacia L. Brown (@slb79 on Twitter) posted cover art of the movie *Like Father* asking "Other Black @netflix users: does your queue do this? Generate posters with the Black cast members on them to try to compel you to watch? This film stars Kristen Bell/Kelsey Grammer and these actors had maaaaybe a 10 cumulative minutes of screen time. 20 lines between them, tops" (stacia l. brown, 2018). Brown's tweet was picked up by various media outlets like *Marie Claire* and *The Guardian*, all reporting that Netflix algorithms were engaging in racism by showing viewers Black actors after observing viewing activity that indicated that they watched "Black content."

I return to the example of Netflix, with which I opened this dissertation because Netflix is an easy target and exemplar of the ways that software companies have frequently received attention for their algorithmic choices and yet managed to escape equivalent critique for their data modeling choices. Complaints about the Netflix algorithm showing too much (or too little) of a type of content, or "customizing" the race of the actors on the content's poster are only possible when there are equivalent supporting data models. What is different about Brown's experience is that the algorithm had access to not only movies watched, but racial data about the actors and the racial composition of the casts of viewed movies.[71] In this case, while the algorithm may have been engaging in racist behavior, it was in part because of the *data model.*

Throughout this project, I have discussed the ways that data models are powerful structuring forces on software systems. In this conclusion, I offer both a summary of the project as well as suggestions for how this work could be used by both researchers and

[71] I am speculating here about how the Netflix algorithm works because it is, like a lot of technology, a closed-source process.

software creators. Data models are an important part of the software development lifecycle, borne as they are out of stakeholder requirements.   Additionally, data models have a cascading structural effect, controlling the shape of databases and thus influencing subsequent data structures used in the written code. The choices that data modelers make are not made in isolation, but within a social and cultural context.

As programmers and tech companies develop awareness of the social and cultural contexts of software production, there are opportunities for small reforms even within the software development lifecycle. Software engineering vanguards can integrate community feedback and participatory design at every stage of the development, including the data modeling portion of the design phase. Drawing on works like *Design Justice* (Costanza-Chock, 2018), teams can stop operating as though their work is separate from their communities and work to fight ideas of the objectivity or neutrality of the software engineering process.   In practice, this might look like open-sourcing portions of the development processes to promote transparency and public conversation about the works.

Within the United States, the infrastructures of software systems are built upon and imbricated within the infrastructures of white supremacy and cisgender normativity. To demonstrate this, I critically analyzed historical data modeling texts and linked several techniques of data modeling to the techniques of white-cisness. That data reification mirrors racial reification is not a coincidence, but a predictable outcome. This project makes an important contribution to demonstrate that mirroring. The takeaway is not that *some* techniques of data modeling uphold white-cisness, but that data models are a site where power manifests. Chapter 2 is also a useful introduction to the emergence of data modeling, and the false ideology of data's natural structure.

As other forms of modeling (like mongodb's document model) grow in popularity, there is perhaps a tendency to reject techniques like reification and generalization because of their connections with white-cis supremacy. I caution against this – moving reactively away from one set of techniques will not separate data models from their cultural valances. Instead, engineering teams using other modeling paradigms have the opportunity to think modeling and race-gender power together. Because many teams will not have the capacity to do this thinking without support, this is a crucial point of intervention for researchers. Those of us with the training to do so can help teams develop new paradigms for data modeling. It is my wish that more researchers develop fluency in both the mechanism of race-gender power structures and the nuances of data modeling techniques and practices. This combination of knowledge is crucial to continue the research into data models.

To move past knowing about data models to seeing the data models within software is an important move towards developing a greater understanding of their power. To that end, I reverse engineered the Lovense Android application. The four steps of reverse engineering that I outlined are available to researchers investigating the social impacts of software. Through the elucidation of those steps, I demonstrated that physical data models are an available artifact, even within some commercial software applications. Once the data model has been uncovered, new analytic opportunities are available.

While many software engineers are familiar with reverse engineering, it is my strong desire for more social science and humanities researchers to integrate reverse engineering into their research practices. I believe that developing the toolset to look within software, not only theoretically, but materially, will greatly enrich critical software studies in a myriad of ways. Importantly, scholarship that uses tools of reverse will be more accessible to software engineers, and will help to move some of these rich critical insights out of the ivory tower and into the office.

As sites of power, and as available artifacts, data models are an important opportunity for intervention for those of us trying to create software for better worlds. I made a data model and proof-of-concept wireframe that, in its design, could mitigate some of the harms that trans people experience when encountering data systems. That this model is possible given our existing tools is evidence that data models are important sites of power --- they are leveraged to reproduce harms, even when there is no technological basis for doing so. Working towards the complete data I made in Chapter 4 involves a lot of choices. Part of my work is to make those choices visible and, through visibility, gestures towards opportunities for intervention. By moving stepwise through the creation of the data model, moving from the common practice of gender as an attribute to gender as one of many gender-relevant entities is an important transformation. The choices I made were based on my own understanding of gender. A different architect (or perhaps even me under different constraints) would produce a different design even based on the same theoretical and ideological foundation.

Engaging with critical technical practice and grounded speculative design is an opportunity for both researchers and engineers.[72] The practice and outcomes of research can feel divorced from the constraints of engineering; software engineers may find academic research inaccessible. However, speculative design work, especially when grounded in a technical practice, can serve as a translational ground at which researchers and engineers cam think together about the potential and limitations of current technology.

[72] Although I address researchers and engineers separately in this conclusion, many of us are both and it is not my desire to imply a separation here.

In the Introduction, I noted that I am accountable to a community of software engineers, a complicated accountability to a community that was created by military-academic industrial capitalism. I am not accountable to software engineering as a field, nor abstract communities of engineers. However, I am accountable to the individual engineers with whom I have worked, trained, and volunteered. Many engineers I know are eager to make change but are themselves trapped within capitalism and claustrophobic ideas that they should be able to do more within their constraints. Despite huge attention towards teaching "ethical engineering," in order to perform as an ethical engineer, one must be quite "heroic" and able to withstand nearly super-human pressures (Basart & Serra, 2013).

Throughout this project, I have emphasized the role that data modelers play as the creators of the model. However, I also want to emphasize that I do not believe that individual engineers have significant power to change the technology on which they work. In part, this is because of the software development lifecycle. The cycles of commercial software production are such that most people on the project (including software engineers, designers, and testers) do not have enough influence to drastically reimagine the project. In my own career, I have had the structural and employment privilege to remove myself from a project with which I had ethical problems. However, I was only able to remove myself because the agency I worked with had another project for me. Had I stayed on this ethically-questionable project, I would have not had any opportunity to impact its outcome. This reflects both the size of the team involved and the capitalist foundation of software production. Within our current methods of production, there is simply no room for most small software companies to push back against problematic customer desires and even less room for individual engineers. I am not here to critique critiques of the ethics of technology production; however it is important to right size the emphasis that individual engineers can exercise on a software project: it is not much.

Although individual engineers may not have much power to make significant changes, in this context, researchers do. Those of us thinking critically about software and society can expand our focus from algorithms to include data structures and data models. Amidst conversations about the inscrutability of algorithms and the unreadability of code, data models remain inherently scrutable. Using the techniques of reverse engineering I outlined in Chapter 3, we can inspect the physical data models of extant software. Using the analysis in Chapter 2, we can begin to think critically about the ways data modeling techniques mirror a variety of systems of power. We can begin to think about using data models as tools of intervention, as I did in Chapter 4. In other words, we can focus on data models as important shaping elements of software.

REFERENCES

Adair, C. (2019). Licensing Citizenship: Anti-Blackness, Identification Documents, and Transgender Studies. *American Quarterly*, 71(2), 569–594. https://doi.org/10.1353/aq.2019.0043

Advantage, n. (n.d.). In OED Online. Oxford University Press. Retrieved February 2, 2022, from http://www.oed.com/view/Entry/2895

Ahmed, S. (2007). A phenomenology of whiteness. *Feminist Theory*, 8(2), 149–168. https://doi.org/10.1177/1464700107078139

Andersson, M. (1994, December). Extracting an entity relationship schema from a relational database through reverse engineering. In *International Conference on Conceptual Modeling* (pp. 403-419). Springer, Berlin, Heidelberg.

Android Open Source Project. (n.d.). Android Open Source Project. Retrieved February 21, 2020, from https://source.android.com/

App Manifest Overview. (n.d.). Android Developers. Retrieved February 21, 2020, from https://developer.android.com/guide/topics/manifest/manifest-intro

App resources overview | Android Developers. (n.d.). Retrieved February 21, 2020, from https://developer.android.com/guide/topics/resources/providing-resources

Auger, J. (2013). Speculative design: Crafting the speculation. *Digital Creativity*, 24(1), 11–35. https://doi.org/10.1080/14626268.2013.767276

Baldwin, G. B., & Shultz, G. P. (1955). Automation: A new dimension to old problems. *Monthly Labor Review*, 165–169.

Baloch, R. (2017). *Ethical Hacking and Penetration Testing Guide*. Auerbach Publications. https://doi.org/10.4324/9781315145891

Bardzell, J., Bardzell, S., & Koefoed Hansen, L. (2015). Immodest Proposals: Research Through Design and Knowledge. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* - CHI '15, 2093–2102. https://doi.org/10.1145/2702123.2702400

Barnett, F., Blas, Z., Cárdenas, M., Gaboury, J., Johnson, J. M., & Rhee, M. (n.d.). QueerOS. Retrieved January 12, 2019, from http://dhdebates.gc.cuny.edu/debates/text/56

Basart, J., & Serra, M. (2013). Engineering Ethics Beyond Engineers' Ethics. *Science and Engineering Ethics*, 19(1), 179–187.

Bauer, R. (2008). Transgressive and transformative gendered sexual practices and white privileges: The case of the dyke/trans BDSM communities. *Women's Studies Quarterly*, 36(3/4), 233–253.

Beauchamp, T. (2019). *Going Stealth: Transgender Politics and U.S. Surveillance Practices*. Duke University Press.

Bender-Baird, K. (2016). Peeing under surveillance: Bathrooms, gender policing, and hate violence. *Gender, Place & Culture*, *23*(7), 983–988. https://doi.org/10.1080/0966369X.2015.1073699

Benjamin, R. (2019b). *Race After Technology: Abolitionist Tools for the New Jim Code* (1st edition). Polity.

Benjamin, R. (Ed.). (2019a). *Captivating technology: Race, carceral technoscience, and liberatory imagination in everyday life*. Duke University Press.

Bergman, S. B. (2009). *The Nearest Exit May Be Behind You.*

Bivens, R. (2017). The gender binary will not be deprogrammed: Ten years of coding gender on Facebook. *New Media & Society*, 19(6), 880–898. https://doi.org/10.1177/1461444815621527

Bizzocchi, J., & Tanenbaum, J. (2011). Well Read: Applying Close Reading Techniques to Gameplay Experiences. In D. Davidson (Ed.), *WellPlayed3.0* (pp. 262–290). ETC Press. http://dl.acm.org/citation.cfm?id=2031858.2031881

Bopp, C., Benjamin, L. M., & Voida, A. (2019). The Coerciveness of the Primary Key: Infrastructure Problems in Human Services Work. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), 51:1-51:26. https://doi.org/10.1145/3359153

Bowker, G. (1994). Information mythology and infrastructure. *Information Acumen: The Understanding and Use of Knowledge in Modern Business*, 1994, 231–247.

Bowker, G. C., Baker, K., Millerand, F., & Ribes, D. (n.d.). *Towards Information Infrastructure Studies: Ways of Knowing in a Networked Environment.* 20.

Boyd, D., & Crawford, K. (2012). Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. *Information, communication & society*, *15*(5), 662-679.

Brock, A. (2020). *Distributed Blackness: African American Cybercultures.* NYU Press.

brown, adrienne maree. (2017). *Emergent Strategy: Shaping Change, Changing Worlds.* AK Press.

Browne, S. (2015). *Dark Matters: On the Surveillance of Blackness.* Duke University Press Books.

CanforaHarman, G., & Di Penta, M. (2007). New Frontiers of Reverse Engineering. *Future of Software Engineering*, 326–341. https://doi.org/10.1109/FOSE.2007.15

Chambers, K., Corbett, J., Keller, C., & Wood, C. (2004). Indigenous Knowledge, Mapping, and GIS: A Diffusion of Innovation Perspective. *Cartographica: The International Journal* *for Geographic Information and Geovisualization*, 39(3), 19–31. https://doi.org/10.3138/N752-N693-180T-N843

Chen, J. N. (2019). *Trans exploits: Trans of color cultures and technologies in movement*. Duke University Press.

Chen, J. N., & cárdenas, micha. (2019). Times to Come. *TSQ: Transgender Studies Quarterly*, 6(4), 472–480. https://doi.org/10.1215/23289252-7771639

Chen, P. P.-S. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1), 9–36.

Chen, Z., Pan, B., & Sun, Y. (2019). A Survey of Software Reverse Engineering Applications. In X. Sun, Z. Pan, & E. Bertino (Eds.), *Artificial Intelligence and Security* (pp. 235–245). Springer International Publishing.

Christin, A. (2020). What Data Can Do: A Typology of Mechanisms. 20.

Chun, W. H. K. (2013). Race and/as Technology, or How to do Things to Race. In *Race after the Internet* (pp. 44–66). Routledge.

Cifor, M., Garcia, P., Cowan, T. L., Rault, J., Sutherland, T., Chan, A. S., Rode, J., Hoffman, A. L., Salehi, N., & Nakamura, L. (n.d.). Feminist Data Manifest-No. Feminist Data Manifest-No. Retrieved November 6, 2021, from https://www.manifestno.com

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. 13(6), 11.

Committee on Innovations in Computing and Communications: Lessons from History, Board, C. S. and T., & Council, N. R. (1999). *Funding a Revolution: Government Support for Computing Research*. National Academies Press.

Costanza-Chock, S. (2018). Design Justice: Towards an Intersectional Feminist Framework for Design Theory and Practice (SSRN Scholarly Paper ID 3189696). *Social Science Research Network*. https://papers.ssrn.com/abstract=3189696

Crenshaw, K. (1989). Demarginalizing the Intersection of Race and Sex: A Black Feminist Critique of Antidiscrimination Doctrine, Feminist Theory and Antiracist Politics. *University of Chicago Legal Forum*, 1989, 139–168.

Crenshaw, K. (1991). Mapping the Margins: Intersectionality, Identity Politics, and Violence Against Women of Color. *Stanford Law Review*, 43(6), 1241–1299.

Currah, P., & Moore, L. J. (2009). "We Won't Know Who You Are": Contesting Sex Designations in New York City Birth Certificates. *Hypatia*, 24(3), 113–135. https://doi.org/10.1111/j.1527-2001.2009.01048.x

D'Ignazio, C., & Klein, L. (2020). *Data Feminism*. MIT Press.

Dahl, O.-J. (2002). The Roots of Object Orientation: The Simula Language. In M. Broy & E. Denert (Eds.), *Software Pioneers: Contributions to Software Engineering* (pp. 78–90). Springer. https://doi.org/10.1007/978-3-642-59412-0_6

Dang, B. (2014). *Practical Reverse Engineering: X86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation* (1 edition). Wiley.

Date, C. J. (2019). *Database design and relational theory: Normal forms and all that jazz*. Apress.

Davis, H. F. (2017). *Beyond Trans: Does Gender Matter?* NYU Press.

Davis, M. (1991). Thinking like an engineer: The place of a code of ethics in the practice of a profession. *Philosophy and Public Affairs*, 20(2), 150–167.

Dolan-Gavitt, B., Hodosh, J., Hulin, P., Leek, T., & Whelan, R. (2015). Repeatable Reverse Engineering with PANDA. *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, 1–11. https://doi.org/10.1145/2843859.2843867

Dourish, P. (2014). No SQL: The Shifting Materialities of Database Technology. *Computational Culture*, 4. http://computationalculture.net/no-sql-the-shifting-materialities-of-database-technology/

Dourish, P. (2016). Algorithms and their others: Algorithmic culture in context. *Big Data & Society*, 3(2), 2053951716665128. https://doi.org/10.1177/2053951716665128

Dow, W. G. (1949). Impacts of electronics on engineering education. *Electrical Engineering*, 68(1), 58–61. https://doi.org/10.1109/EE.1949.6444563

Drucker, J. (2011). Humanities Approaches to Graphical Display. *Digital Humanities Quarterly*, 005(1).

Dunbar-Ortiz, R. (2014). *An indigenous peoples' history of the United States* (Vol. 3). Beacon Press.

Duster, T. (2005). Race and reification in science. *Science*, 307, 1050–1051.

Dwivedi, A. K., Rath, S. K., Satapathy, S. M., Chakravarthy, L. S., & Rao, P. K. S. (2018, September). Applying Reverse Engineering Techniques to Analyze Design Patterns in Source Code. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1398-1404). IEEE.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering* (1 edition). Wiley.

Enke, A. F. (2012). The education of little cis: Cisgender and the discipline of opposing bodies. In *Transfeminist perspectives in and beyond transgender and gender studies* (pp. 60–77).

Enns, P. K., Yi, Y., Comfort, M., Goldman, A. W., Lee, H., Muller, C., Wakefield, S., Wang, E. A., & Wildeman, C. (2019). What Percentage of Americans Have Ever Had a Family Member Incarcerated?: Evidence from the Family History of Incarceration

Survey (FamHIS). Socius, 5, 2378023119829332. https://doi.org/10.1177/2378023119829332

Eubanks, V. (2018). *Automating Inequality: How High-Tech Tools Profile, Police, and Punish the Poor.* St. Martin's Press.

Faustino, M. J. (2018). Rebooting an Old Script by New Means: Teledildonics—The Technological Return to the 'Coital Imperative.' *Sexuality & Culture*, 22(1), 243–257. https://doi.org/10.1007/s12119-017-9463-5

Ferguson, R. A. (2003). *Aberrations In Black: Toward A Queer Of Color Critique* (First edition edition). Univ Of Minnesota Press.

Fischer, M. (2019). *Terrorizing gender: Transgender visibility and the surveillance practices of the US security state.* U of Nebraska Press.

Fiske, J. (1998). Surveilling the city: Whiteness, the black man and democratic totalitarianism. *Theory, Culture & Society,* 15(2), 67–88.

Frankel, F., & DePace, A. H. (2012). Visual Strategies: A Practical Guide to Graphics for Scientists & Engineers. *Yale University Press.*

Gaboury, J. (2018). Becoming NULL: Queer relations in the excluded middle. Women & Performance: *A Journal of Feminist Theory,* 28(2), 143–158. https://doi.org/10.1080/0740770X.2018.1473986

Garner, S. (2007). *Whiteness: An introduction.* Routledge.

Gates, K. (2014). *Can Computers Be Racist?* 5–18.

Gehl, R. (2015). Critical reverse engineering: The case of Twitter and TalkOpen. *Compromised Data: From Social Media to Big Data*, 147.

Gehl, R. W., & Bell, S. A. (2012). Heterogeneous Software Engineering: Garmisch 1968, Microsoft Vista, and a Methodology for Software Studies. Computational Culture, 2. http://computationalculture.net/heterogeneous-software-engineering-garmisch-1968-microsoft-vista-and-a-methodology-for-software-studies/

Genosko, G., & Thompson, S. (2006). Tense theory: The temporalities of surveillance. In *Theorizing Surveillance* (pp. 137–152). Willan.

Gitelman, L. (2013). *Raw data is an oxymoron.* MIT Press.

Green, K. M., & Bey, M. (2017). Where Black Feminist Thought and Trans* Feminism Meet: A Conversation. *Souls*, 19(4), 438–454. https://doi.org/10.1080/10999949.2018.1434365

Green, N., & Zurawski, N. (2015). Surveillance and Ethnography: Researching Surveillance as Everyday Life. *Surveillance & Society*, *13*(1), 27–43.

Grodzinsky, F. S., Miller, K., & Wolf, M. J. (2012). Moral Responsibility for Computing Artifacts: "the Rules" and Issues of Trust. *SIGCAS Comput. Soc.*, 42(2), 15–25. https://doi.org/10.1145/2422509.2422511

Haigh, T. (2009). How Data Got its Base: Information Storage Software in the 1950s and 1960s. *IEEE Annals of the History of Computing*, 31(4), 6–25. https://doi.org/10.1109/MAHC.2009.123

Halberstam, J. (1998). Transgender butch: Butch/FTM border wars and the masculine continuum. *GLQ: A Journal of Lesbian and Gay Studies*, 4(2), 287–310.

Hale, C. J. (1997). Leatherdyke Boys and Their Daddies: How to Have Sex without Women or Men. *Social Text*, 52/53, 223–236. JSTOR. https://doi.org/10.2307/466741

Hamm, M. J. (2014). *Wireframing Essentials* (Illustrated edition). Packt Publishing.

Harris, C. I. (2006). Whiteness as Property (SSRN Scholarly Paper ID 927850). *Social Science Research Network*. https://papers.ssrn.com/abstract=927850

Harris, D. R., Reubenstein, H. B., & Yeh, A. S. (1995). Reverse Engineering to the Architectural Level. *1995 17th International Conference on Software Engineering,* 186–186. https://doi.org/10.1145/225014.225032

Hauser, C. (2021, October 27). U.S. Issues First Passport With 'X' Gender Marker. *The New York Times*. https://www.nytimes.com/2021/10/27/us/us-first-passport-gender-x.html

Hoffmann, A. L. (2018). Data Violence and How Bad Engineering Choices Can Damage Society. *Medium*. https://medium.com/s/story/data-violence-and-how-bad-engineering-choices-can-damage-society-39e44150e1d4

Hoffmann, A. L. (2019). Where fairness fails: Data, algorithms, and the limits of antidiscrimination discourse. *Information, Communication & Society*, 22(7), 900–915. https://doi.org/10.1080/1369118X.2019.1573912

Hoffmann, A. L. (2020). Terms of Inclusion: Data, Discourse, Violence. *New Media & Society*, preprint.

Holmevik, J. R. (1994). Compiling SIMULA: A historical study of technological genesis. *IEEE Annals of the History of Computing*, 16(4), 25–37. https://doi.org/10.1109/85.329756

Hoos, I. R. (1960). The sociological impact of automation in the office. *Management Science*, 2, 10–19.

Hughes, L. (2018). Wronging the right-body narrative: On the universality of gender uncertainty. In *Current critical debates in the field of transsexual studies* (pp. 181–193). Routledge.

Hujainah, F., Bakar, R. B. A., Abdulgabber, M. A., & Zamli, K. Z. (2018). Software Requirements Prioritisation: A Systematic Literature Review on Significance, Stakeholders, Techniques and Challenges. *IEEE Access*, 6, 71497–71523. https://doi.org/10.1109/ACCESS.2018.2881755

Iliadis, A., & Russo, F. (2016). Critical data studies: An introduction. *Big Data & Society*, *3*(2), 2053951716674238.

Institute for Policy Studies. (n.d.). Racial Economic Inequality. Inequality.Org. Retrieved October 16, 2019, from https://inequality.org/facts/racial-inequality/

Johnson, J. M. (2018). Markup Bodies: Black [Life] Studies and Slavery [Death] Studies at the Digital Crossroads. *Social Text*, 36(4 (137)), 57–79. https://doi.org/10.1215/01642472-7145658

Kerssens, N. (2018). The Database 'Revolution': The Technological and Cultural Origins of the Big-data-based Mindset in American Management, 1970s–1980s. *TMG Journal for Media History,* 21(2), 7–29. https://doi.org/10.18146/2213-7653.2018.364

Kitchin, R., & Lauriault, T. (2014). Towards critical data studies: Charting and unpacking data assemblages and their work.

Kitchin, R. (2017). Thinking critically about and researching algorithms. *Information, Communication & Society*, *20*(1), 14–29. https://doi.org/10.1080/1369118X.2016.1154087

Lanza, M., & Ducasse, S. (2003). Polymetric views—A lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9), 782–795. https://doi.org/10.1109/TSE.2003.1232284

Lau, J. R. (2016). Between the Times: Trans-Temporality, and Historical Representation. UCLA.

Lavizzo-Mourey, R., & Williams, D. (2016, April 14). Being Black Is Bad for Your Health. *US News & World Report*. https://www.usnews.com/opinion/blogs/policy-dose/articles/2016-04-14/theres-a-huge-health-equity-gap-between-whites-and-minorities

Levitt, H. M., & Hiestand, K. R. (2004). A Quest for Authenticity: Contemporary Butch Gender. *Sex Roles*, 50(9/10), 605–621. https://doi.org/10.1023/B:SERS.0000027565.59109.80

Levy, K. E. C. (2014). Intimate Surveillance. *Idaho Law Review*, *3*, 679–694.

Liberati, N. (2017). Teledildonics and New Ways of "Being in Touch": A Phenomenological Analysis of the Use of Haptic Devices for Intimate Relations. *Science and Engineering Ethics*, 23(3), 801–823. https://doi.org/10.1007/s11948-016-9827-5

Lin, Z., Zhang, X., & Xu, D. (2010). Reverse Engineering Input Syntactic Structure from Program Execution and Its Applications. *IEEE Transactions on Software Engineering*, 36(5), 688–703. https://doi.org/10.1109/TSE.2009.54

Lodge, M., & Mennicken, A. (2017). The Importance of Regulation of and by Algorithm. *Algorithmic Regulation*, 2.

Lovense Bluetooth Sex Toys—History of Innovative Sex Tech. (n.d.). Lovense. Retrieved February 6, 2020, from https://www.lovense.com/sextoys/about-us

Lovense Sex Toys for Long-Distance Love. (n.d.). Lovense. Retrieved April 19, 2021, from https://www.lovense.com/compare

Lovense. (n.d.). Max 2 by Lovense. The Best Wireless Male Masturbator. Lovense. Retrieved February 6, 2020, from https://www.lovense.com/male-masturbators

Lu, J. H., & Steele, C. K. (2019). 'Joy is resistance': Cross-platform resilience and (re)invention of Black oral culture online. *Information, Communication & Society*, 22(6), 823–837. https://doi.org/10.1080/1369118X.2019.1575449

Lubis, J. H., & Zamzami, E. M. (2020, June). Relational database reconstruction from SQL to Entity Relational Diagrams. In *Journal of Physics: Conference Series* (Vol. 1566, No. 1, p. 012072). IOP Publishing.

Lyon, D., Waldo, J., Lin, H. S., & Millett, L. I. (2007). A Short History of Surveillance and Privacy in the United States. *Engaging Privacy and Information Technology in a Digital Age.*

Maalsen, S., & Sadowski, J. (2019). The Smart Home on FIRE: Amplifying and Accelerating Domestic Surveillance. *Surveillance & Society*, 17(1/2), 118–124. https://doi.org/10.24908/ss.v17i1/2.12925

Malpass, M. (2013). Between Wit and Reason: Defining Associative, Speculative, and Critical Design in Practice. *Design and Culture*, 5(3), 333–356. https://doi.org/10.2752/175470813X13705953612200

Malpass, M. (2017). *Critical design in context: History, theory, and practices*. Bloomsbury Publishing.

Mascarenhas, M. (2018). White Space and Dark Matter: Prying Open the Black Box of STS. *Science, Technology, & Human Values*, 43(2), 151–170. https://doi.org/10.1177/0162243918754672

McGuirk, J. (2015). Honeywell, I'm Home! The Internet of Things and the New Domestic Landscape. *E-Flux*, 64. https://www.e-flux.com/journal/64/60855/honeywell-i-m-home-the-internet-of-things-and-the-new-domestic-landscape/

McPherson, T. (2013). U.S. Operating Systems at Mid-Century: The Intertwining of Race and UNIX. In L. Nakamura & Peter Chow-White (Eds.), *Race After the Internet*. Routledge.

McPherson, T. (2018). *Feminist in a software lab: Difference+ design*. Harvard University Press.

Meyerowitz, J. (2002*). How Sex Changed: A History of Transsexuality in the United States*. Harvard University Press. https://doi.org/10.2307/j.ctv1c7zfrv

Mills, C. W. (2004). Racial exploitation and the wages of whiteness. In *What white looks like* (pp. 41–70). Routledge.

Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S., & Floridi, L. (2016). The ethics of algorithms: Mapping the debate. *Big Data & Society*, 3(2), 2053951716679679. https://doi.org/10.1177/2053951716679679

Monahan, T. (2010). *Surveillance in the Time of Insecurity*. Rutgers University Press.

Muñoz, J. E. (2009). Introduction. In *Cruising Utopia: The Then and There of Queer Futurity*. New York University Press. http://ebookcentral.proquest.com/lib/asulib-ebooks/detail.action?docID=865693

Mylopoulos, J. (1980). A perspective for research on conceptual modelling. *Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling*, 167–170.

N.P. v Standard Innovation Corp, 16 CV 8655 (Illinois Eastern August 15, 2017).

Neilson, B. (2020). The Reverse of Engineering. *South Atlantic Quarterly*, 119(1), 75–93. https://doi.org/10.1215/00382876-8007665

Nelson, A. (2016). *Social Life of DNA*. Beacon Press.

Noble, D. F. (1979). *America by Design: Science, Technology, and the Rise of Corporate Capitalism*. Oxford University Press.

Nora by Lovense. The Original Bluetooth Rabbit Vibrator! (n.d.). Lovense. Retrieved February 6, 2020, from https://www.lovense.com/rabbit-vibrator

Olivé, A. (2007). *Conceptual modeling of information systems*. Springer.

Owen, T. (2015). The Violence of Algorithms. Foreign Affairs. https://www.foreignaffairs.com/articles/2015-05-25/violence-algorithms

Painter, N. I. (2011). *The History of White People*. W. W. Norton & Company.

Parikka, J. (2013). *What is Media Archaeology?* Wiley & Sons.

Pen Test Partners. (n.d.). Smart male chastity lock cock-up | Pen Test Partners. Retrieved August 18, 2021, from https://www.pentestpartners.com/security-blog/smart-male-chastity-lock-cock-up/?=october-5-2020

Pinch, T. J., & Bijker, W. E. (1984). The Social Construction of Facts and Artefacts: Or How the Sociology of Science and the Sociology of Technology might Benefit Each Other. *Social Studies of Science,* 14(3), 399–441. https://doi.org/10.1177/030631284014003004

Premerlani, W. J., & Blaha, M. R. (1993, May). An approach for reverse engineering of relational databases. In *[1993] Proceedings Working Conference on Reverse Engineering* (pp. 151-160). IEEE.

Prosser, J. (1998). *Second Skins: The Body Narratives of Transsexuality*, 288. Columbia University Press.

Qiui. (n.d.). Cellmate Chastity Cage. QIUI. Retrieved August 18, 2021, from https://www.qiui.store/product-page/cellmate-chastity-cage

Quinan, C. (2017). Gender (in) securities: Surveillance and transgender bodies in a post-9/11 era of neoliberalism. *Security/Mobility,* 153.

Rana, J. (2016). The Racial Infrastructure of the Terror-Industrial Complex. *Social Text,* 34(4 (129)), 111–138. https://doi.org/10.1215/01642472-3680894

Rapoport, M. (2012). The Home Under Surveillance: A Tripartite Assemblage. *Surveillance & Society*, *10*(3/4), 320–333.

Razali, R., & Anwar, F. (2011). Selecting the right stakeholders for requirements elicitation: A systematic approach. *Journal of Theoretical and Applied Information Technology*, 33(2), 250–257.

Rekoff, M. G. (1985). On reverse engineering. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(2), 244–252. https://doi.org/10.1109/TSMC.1985.6313354

Richterich, A. (2018). *The Big Data Agenda: Data Ethics and Critical Data Studies*. University of Westminster Press.

Roberts, D. E. (2009). Race, gender, and genetic technologies: A new reproductive dystopia? *Signs: Journal of Women in Culture and Society*, 34(4), 783–804.

Roediger, D. R. & others. (1999). *The wages of whiteness: Race and the making of the American working class*. Verso.

Rovner, J. (n.d.). Racial Disparities in Youth Commitments and Arrests. The Sentencing Project. Retrieved January 31, 2018, from http://www.sentencingproject.org/publications/racial-disparities-in-youth-commitments-and-arrests/

Rubin, G. (2013). Of catamites and kings: Reflections on butch, gender, and boundaries. In *The transgender studies reader* (pp. 487–497). Routledge.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes,* 35(3), 8–13. https://doi.org/10.1145/1764810.1764814

Sandvig, C., Hamilton, K., Karahalios, K., & Langbort, C. (2014). Auditing Algorithms: Research Methods for Detecting Discrimination on Internet Platforms. *International Communication Association,* 1–20.

Scott, J. C. (1998). *Seeing like a state: How certain schemes to improve the human condition have failed.* Yale University Press.

Sears, C. (2015). Arresting Dress: Cross-Dressing, Law, and Fascination in Nineteenth-Century San Francisco. In *Arresting Dress*. Duke University Press. https://doi.org/10.1515/9780822376194

Shi, N., & Olsson, R. A. (2006, September). Reverse engineering of design patterns from java source code. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)* (pp. 123-134). IEEE.

Simsion, G. C., & Witt, G. C. (2005). *Data modeling essentials* (3rd ed). Morgan Kaufmann Publishers.

Simula, B. L. (2021). Introduction to the special issue: BDSM Studies. *Sexualities*, 1363460721993039.

Skidmore, E. (2011). Constructing the "Good Transsexual": Christine Jorgensen, Whiteness, and Heteronormativity in the Mid-Twentieth-Century Press. *Feminist Studies*, 37(2), 270–300.

Smith, J. M., & Smith, D. C. (1977). Database abstractions: Aggregation. *Communications of the ACM*, 20(6), 405–413.

Smith, W. (n.d.). *Everyday Data Structures*. 393.

Snorton, C. R. (2017). *Black on Both Sides: A Racial History of Trans Identity (3rd ed. edition)*. Univ Of Minnesota Press.

Sommerville, I. (2015). *Software engineering* (10th ed.). Addison-Wesley.

Spade, D. (2008). Documenting Gender. *Hastings Law Journal*, 59, 113.

Spade, D. (2011). *Normal life: Administrative violence, critical trans politics, and the limits of law.* South End Press.

Srivastava, M. (2015). A View of Reverse Engineering. 2395, 4.

stacia l. brown. (2018, October 18). Other Black @netflix users: Does your queue do this? Generate posters with the Black cast members on them to try to compel you to watch? This film stars Kristen Bell/Kelsey Grammer and these actors had maaaaybe a 10 cumulative minutes of screen time. 20 lines between them, tops. Https://t.co/Sj7rD8wfOS [Tweet]. @slb79. https://twitter.com/slb79/status/1052776984231718912

Stanley, E. A. (2014). Gender self-determination. *Transgender Studies Quarterly,* 1(1–2), 89–91.

Stanley, E. A. (2021). *Atmospheres of violence: Structuring antagonism and the Trans/Queer ungovernable.* Duke University Press.

Star, S. L. (1999). The Ethnography of Infrastructure. *American Behavioral Scientist,* 43(3), 377–391. https://doi.org/10.1177/00027649921955326

Stevens, N. L., Hoffmann, A. L., & Florini, S. (2021). The Unremarked Optimum: Whiteness, Optimization, and Control in The Database Revolution. *Review of Communication.*

Stone, S. (1992). The Empire Strikes Back: A Posttranssexual Manifesto. *Camera Obscura: Feminism, Culture, and Media Studies*, 10(2 (29)), 150–176. https://doi.org/10.1215/02705346-10-2_29-150

Stryker, S., & Aizura, A. (Eds.). (2013). *The Transgender Studies Reader 2* (1 edition). Routledge.

Stryker, S., & Currah, P. (2014). Postposttranssexual: Key concepts for a twenty-first-century transgender studies. *TSQ: Transgender Studies Quarterly*, 1–1.

Sue, D. W. (2006). The Invisible Whiteness of Being: Whiteness, White Supremacy, White Privilege, and Racism. In *Addressing racism: Facilitating cultural competence in mental health and educational settings* (pp. 15–30). John Wiley & Sons Inc.

Thomer, A. K., & Wickett, K. M. (2020). Relational data paradigms: What do we learn by taking the materiality of databases seriously?: *Big Data & Society.* https://doi.org/10.1177/2053951720934838

Tran, D. (2003). An Excerpt from Call Me by My Proper Name: Legal Surveillance and Discipline of Transgender Body and Identity. *Women's Rts. L. Rep.*, *25*, 219.

Tuck, E., & Yang, K. W. (2012). Decolonization is not a metaphor. *Decolonization: Indigeneity, Education & Society*, 1(1).

Tumbleson, C., & Wiśniewski, R. (n.d.). Apktool—A tool for reverse engineering 3rd party, closed, binary Android apps. Retrieved December 27, 2021, from https://ibotpeaches.github.io/Apktool/

Udupa, S. K., Debray, S. K., & Madou, M. (2005, November). Deobfuscation: Reverse engineering obfuscated code. In *12th Working Conference on Reverse Engineering (WCRE'05)* (pp. 10-pp). IEEE.

Wachter-Boettcher, S. (2017). *Technically Wrong: Sexist Apps, Biased Algorithms, and Other Threats of Toxic Tech* (1 edition). W. W. Norton & Company.

Wajcman, J. (2004). *TechnoFeminism* (1 edition). Polity.

Wamsley, L. (2017, June 16). Oregon Adds A New Gender Option To Its Driver's Licenses: X. NPR. https://www.npr.org/sections/thetwo-way/2017/06/16/533207483/oregon-adds-a-new-gender-option-to-its-driver-s-licenses-x

West, M. (2011). *Developing high quality data models*. Morgan Kaufmann.

Whittaker, Z. (n.d.). This smart vibrator can be "easily" hacked and remotely controlled by anyone. ZDNet. Retrieved August 18, 2021, from https://www.zdnet.com/article/this-smart-vibrator-can-be-easily-hacked-and-remotely-controlled-by-anyone/

Winner, L. (1986). Do Artifacts Have Politics. In *The Whale and the Reactor: The Search for Limits in an Age of High Technology* (pp. 19–39). Chicago University Press.

Wirth, N. (1976). *Algorithms + Data Structures = Programs* (1st edition). Prentice Hall.

Woodard, J. B., & Mastin, T. (2005). Black Womanhood: Essence and its Treatment of Stereotypical Images of Black Women. *Journal of Black Studies*, 36(2), 264–281. https://doi.org/10.1177/0021934704273152

Wynn, M. A. (2018). Categorizing the Security and Privacy of "Internet of Things" Devices [Thesis]. https://utd-ir.tdl.org/handle/10735.1/5932

Xu, J. (2019, July 1). Algorithmic Solutions to Algorithmic Bias: A Technical Guide. Medium. https://towardsdatascience.com/algorithmic-solutions-to-algorithmic-bias-aef59eaf6565

APPENDIX A

DATABASE CREATION SQL

The SQL code below is extracted from the annotations that I found in the Lovense codebase and what I used to generate the data model in Chapter 3.

```
CREATE TABLE IF NOT EXISTS `tb_commun_message` (
   `data` TEXT NULL,
   `dataBean` TEXT NULL,
   `from` TEXT NULL,
   `sendStatus` INT NULL,
   `sendType` INT NULL,
   `status` TEXT NULL,
   `f2796to` TEXT NULL,
   `type` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_account_setting` (
   `isTip` BOOLEAN NULL,
   `userId` TEXT NULL,
   `version` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_alarm` (
   `friendEmail` TEXT NULL,
   `message` TEXT NULL,
   `missSwitchOn` TEXT NULL,
   `notifySwitchOn` TEXT NULL,
   `ownerEmail` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_alarm_lists` (
   `accept` INT NULL,
   `alarmTitle` TEXT NULL,
   `bean` TEXT NULL,
   `dates` TEXT NULL,
   `duration` INT NULL,
   `frequency` TEXT NULL,
   `friendEmail` TEXT NULL,
   `haveSnoozeCount` INT NULL,
   `isSelected` INT NULL,
   `nextTime` TEXT NULL,
   `notiType` TEXT NULL,
   `notify` INT NULL,
   `ownerEmail` TEXT NULL,
   `patternId` TEXT NULL,
   `receiveAlarmId` TEXT NULL,
   `receiveFlag` INT NULL,
   `reveiveFilePath` TEXT NULL,
   `ringTime` TEXT NULL,
   `sendTime` TEXT NULL,
   `snoozeCount` INT NULL,
   `snoozeDuration` INT NULL,
   `soundFileath` TEXT NULL,
   `soundurl` TEXT NULL,
   `time` TEXT NULL,
   `version` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_alarm_pattern` (
   `alarmUrl` TEXT NULL,
   `author` TEXT NULL,
   `checkMd5` BOOLEAN NULL,
   `creator` TEXT NULL,
   `data` TEXT NULL,
   `email` TEXT NULL,
   `isExist` BOOLEAN NULL,
   `likeCount` INT NULL,
   `name` TEXT NULL,
   `path` TEXT NULL,
   `shared` BOOLEAN NULL,
   `sortId` INT NULL,
   `text` TEXT NULL,
   `timer` TEXT NULL,
   `toyFunc` TEXT NULL,
   `toyTag` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_back_work` (
   `owner` TEXT NULL,
   `staticParams` TEXT NULL,
   `status` TEXT NULL,
   `targetEmail` TEXT NULL,
   `workId` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_emoji_favorite` (
   `emojiId` TEXT NULL,
   `owner` TEXT NULL,
   `path` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_hot_point` (
   `appVersionCode` INT NULL,
   `isPressed` BOOLEAN NULL,
   `owner` TEXT NULL,
   `resId` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_log_type` (
   `content` TEXT NULL,
   `logNo` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_msg_unread` (
   `friendJid` TEXT NULL,
   `ownerJid` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_msg_hide` (
   `friendJid` TEXT NULL,
   `ownerJid` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_media_pattern` (
   `creator` TEXT NULL,
   `friend` TEXT NULL,
   `pattern` INT NULL,
   `patternId` TEXT NULL,
   `patternJson` TEXT NULL,
   `states` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;



CREATE TABLE IF NOT EXISTS `tb_merger_music` (
   `artist` TEXT NULL,
   `data` TEXT NULL,
   `duration` INT NULL,
   `email` TEXT NULL,
   `imageUrl` TEXT NULL,
   `musicType` INT NULL,
   `playlistId` TEXT NULL,
   `songIdOrUri` TEXT NULL,
   `title` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_music` (
   `album` TEXT NULL,
   `albumId` FLOAT NULL,
   `artist` TEXT NULL,
   `cover` BLOB NULL,
   `data` TEXT NULL,
   `duration` INT NULL,
   `imageUrl` TEXT NULL,
   `isFavorite` BOOLEAN NULL,
   `mergerType` INT NULL,
   `musicType` INT NULL,
   `songId` FLOAT NULL,
   `title` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_music_playlist` (
   `cover` TEXT NULL,
   `createFromLocalDB` INT NULL,
   `email` TEXT NULL,
   `isStreamMusic` BOOLEAN NULL,
   `name` TEXT NULL,
   `notice` TEXT NULL,
   `sortId` INT NULL,
   `total` INT NULL,
   `tracksUrl` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_music_playlist_items` (
   `email` TEXT NULL,
   `music` BLOB NULL,
   `musicType` INT NULL,
   `playlistId` TEXT NULL,
   `songId` TEXT NULL,
   `sortId` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_playlist_items` (
   `email` TEXT NULL,
   `patternId` TEXT NULL,
   `playlistId` TEXT NULL,
   `sortId` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_playlist` (
   `email` TEXT NULL,
   `name` TEXT NULL,
   `sortId` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

/** patterns **/
CREATE TABLE IF NOT EXISTS `tb_pattern` (
   `download` TEXT NULL,
   `author` TEXT NULL,
   `cdnPath` TEXT NULL,
   `checkMd5` BOOLEAN NULL,
   `creator` TEXT NULL,
   `data` TEXT NULL,
   `email` TEXT NULL,
   `favorite` BOOLEAN NULL,
   `isAnony` INT NULL,
   `isExist` BOOLEAN NULL,
   `isOnline` BOOLEAN NULL,
   `isShowFormatToast` BOOLEAN NULL,
   `likeCount` INT NULL,
   `name` TEXT NULL,
   `path` TEXT NULL,
   `shared` BOOLEAN NULL,
   `sortId` INT NULL,
   `status` TEXT NULL,
   `text` TEXT NULL,
   `timer` TEXT NULL,
   `toyFunc` TEXT NULL,
   `toyTag` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_program_pattern` (
   `author` TEXT NULL,
   `btAddress` TEXT NULL,
   `data` TEXT NULL,
   `email` TEXT NULL,
   `index` TEXT NULL,
   `name` TEXT NULL,
   `timer` TEXT NULL,
   `uploadToToyTotalComman` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_re_call` (
   `canBe` INT NULL,
   `flag` INT NULL,
   `msgId` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_setting` (
   `acceptPrivacyLogs` BOOLEAN NULL,
   `autoLastLevel` BOOLEAN NULL,
   `autoSwithOff` BOOLEAN NULL,
   `playMusicModel` INT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_switch_values` (
   `key` INT NULL,
   `owner` TEXT NULL,
   `values` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_test_values` (
   `key` TEXT NULL,
   `type` TEXT NULL,
   `value` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_toy` (
   `expandName2` TEXT NULL,
   `aColor` INT NULL,
   `address` TEXT NULL,
   `agString` TEXT NULL,
   `aiString` TEXT NULL,
   `batchId` TEXT NULL,
   `battery` INT NULL,
   `bindType` INT NULL,
   `canRssi` BOOLEAN NULL,
   `connectedTime` FLOAT NULL,
   `defineRename` TEXT NULL,
   `deviceId` TEXT NULL,
   `deviceName` TEXT NULL,
   `deviceType` TEXT NULL,
   `disConnectType` INT NULL,
   `email` TEXT NULL,
   `getCheckBindToyErrorTime` FLOAT NULL,
   `getDfuErrorTime` FLOAT NULL,
   `isCheckBindToy` INT NULL,
   `isDfuEnd` INT NULL,
   `isSelect` INT NULL,
   `isTransfer` BOOLEAN NULL,
   `led` INT NULL,
   `ledSetting` INT NULL,
   `name` TEXT NULL,
   `requestConnectingIndex` INT NULL,
   `rquestConnectTime` FLOAT NULL,
   `rssi` INT NULL,
   `simpleToy` INT NULL,
   `status` INT NULL,
   `type` TEXT NULL,
   `version` INT NULL)
```

```
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;;

CREATE TABLE IF NOT EXISTS `tb_toy_rename` (
   `address` TEXT NULL,
   `email` TEXT NULL,
   `name` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_toy_type` (
   `aColor` INT NULL,
   `address` TEXT NULL,
   `autoLightOn` BOOLEAN NULL,
   `type` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_user` (
   `acceptVideoRequestJid` TEXT NULL,
   `addMessage` BOOLEAN NULL,
   `addTime` FLOAT NULL,
   `autoAccepts` TEXT NULL,
   `avatar` TEXT NULL,
   `avatarBitmapUrl` TEXT NULL,
   `chatType` TEXT NULL,
   `currentControlType` TEXT NULL,
   `deviceAppVersion` TEXT NULL,
   `deviceToken` TEXT NULL,
   `deviceType` TEXT NULL,
   `emailAndPassMd5` TEXT NULL,
   `fagree` INT NULL,
   `friendGTMTime` TEXT NULL,
   `friendType` INT NULL,
   `gender` TEXT NULL,
   `iagree` INT NULL,
   `liveFriendId` TEXT NULL,
   `liveStatus` INT NULL,
   `logNotices` TEXT NULL,
   `moodMessage` TEXT NULL,
   `name` TEXT NULL,
   `online` BOOLEAN NULL,
   `password` TEXT NULL,
   `remark` TEXT NULL,
   `remotePlatform` TEXT NULL,
   `remoteVersion` TEXT NULL,
   `setTop` FLOAT NULL,
   `tempAvatar` TEXT NULL,
   `tempName` TEXT NULL,
   `tempViewType` INT NULL,
   `toyDeviceId` TEXT NULL,
   `toyName` TEXT NULL,
   `toyStatus` TEXT NULL,
   `uid` TEXT NULL,
   `userCode` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;

CREATE TABLE IF NOT EXISTS `tb_user_setting` (
   `addTime` FLOAT NULL,
   `audioVibration` BOOLEAN NULL,
   `autoAccept` BOOLEAN NULL,
   `autoPlayAlarm` BOOLEAN NULL,
   `autoPlayPattern` BOOLEAN NULL,
   `friendUserId` TEXT NULL,
   `friendsRequestClick` BOOLEAN NULL,
   `logNotify` BOOLEAN NULL,
   `messageVibration` BOOLEAN NULL,
   `passiveConfig` TEXT NULL,
   `userId` TEXT NULL)
   ENGINE = InnoDB
   DEFAULT CHARSET = utf8mb4
   COLLATE = utf8mb4_general_ci;
```