**What is a database?** – An external storage system (separate from our code) that can be queried to retrieve, update and insert data. The structure of a database is dependent on the type of database and our application. **Relational Database Model** – Model that considers the presentation of data to the programmer, not about its storage on disk. Storage strategy is separate from the programmer view. Data is presented as a series of relations and data is manipulated using SQL. **Database Management System** – The system responsible for making sure our data is persistent, accessed efficiently, making sure queries are responded to, etc. The DBMS is the only thing interacting with the disk (not us, not the program). **Relation** – A relation is the structure of a data set. For example R(A,B,C). All instances of a particular relation have the same structure (Always have A,B,C) and not two instances can be the same. **Instance** – a row in our relation. **Attribute** – a column in a relation. **Schema** – the structure of all of the relations in a database. **Attribute Rules** – Must be elementary types, no compound data structures, every attribute must only be one type. **Domain** – The subject matter of a database. **Key** – set of attributes that uniquely identify instances within a relation. **Relationships between Relations** – One-to-one One-to-many Many-to-many. **Creating a table in SQL** - CREATE TABLE *tablename*(attribute type, attribute type, …), For example: CREATE TABLE Movies(title VARCHAR(25), year INT, length INT, genre VARCHAR(15)).

**Modifying a schema** –

DROP TABLE Movies

*There is no coming back from this unless you have a backup.*

ALTER TABLE MOVIES ADD studioName CHAR(30)

ALTER TABLE MOVIES DROP year

*Making a default for a column*

ALTER TABLE MOVIES ADD studioName CHAR(30) DEFAULT ‘Independent’

CREATE TABLE Star(name VARCHAR(20) PRIMARY KEY, address VARCHAR(255), date\_of\_birth DATE)

CREATE TABLE Movies(title VARCHAR(25), …, PRIMARY KEY(attribute\_1, attribute\_2)

INSERT INTO Movies(title, year, genre) VALUES(‘Dark Knight’, 2008, action)

**Relational Operations –**

SELECT – for projections

WHERE – for selections

FROM – which relation are we operating on

*Basic function:*

SELECT \* FROM Movies

SELECT genre FROM Movies

SELECT \* FROM Movies WHERE year == 1998

SELECT genre FROM Movies WHERE year == 1998

**Functional Dependency** – If two instances of arelation agree on a set of attributes, they must also agree on some other set of attributes. Basically, each functional dependency is a key. **Normal Forms –** ‘Levels’ of proper design. *1st Normal Form*: BCNF -> each functional dependency between attributes represent a key in the relation. If when we try to decompose our functional dependencies to BCNF we lose a functional dependency, we move to *3rd Normal Form*. This means we have two options: we can enforce the functional dependency we lost (can make a mess) or relax the BCNF constraint to -> each attribute on the right is a member of any other key. *4th Normal Form* -> if we know one column, there are a set of options for the other columns, which essentially means that there are no functional dependencies.

**Steps to make a good design**

1. Locate/Define FDs
2. Decompose into BCNF
3. Check for FD loss
   1. If lost, try 3rd Normal Form starting from Step 1
4. Check for multivalued dependencies
   1. Decompose to 4th Normal form if needed

**SQL Deep Dive** – Cross product is the most fundamental operation but directly is the least useful. Defined as each instance in T is paired with each instance in S. Doesn’t necessarily create meaningful data. Can result in a massive data set. Joins calculate the cross product without extraneous data. There are two kinds of Joins: natural joins were the resulting rows must agree on attributes, and a theta join where we are specifying the condition for which the result set is based on.

SELECT \* FROM Orders, PC WHERE Orders.Model# = PC.Model# AND Date < ‘2014/1/1’ 🡪 Cross Product

SELECT \* FROM Orders NATURAL JOIN PC WHERE Date < ‘2014/1/1’ 🡪 Natural Join

SELECT \* FROM Orders JOIN PC ON Orders.Model# = PC.Model# AND Date < ‘2014/1/1’ 🡪 Theta Join

To specify that an attribute must not be null in a relation, we can specify that when we create the table:

CREATE TABLE Persons {

ID int NOT NULL,

LastName varchar(255) NOT NULL,

…

}

Inner Join means that the two things being joined must match completely. Left join will return the inner join plus results from the *left* that did not have matches on the right. Full outer join includes an inner join, a left join, and what is essentially a right join.

Order By acts as a sorting operator that will group results by whatever column we specify. The defulat order is ascending but we can include DESC to have descending order.

*Renaming* - SELECT StarsIn.starName as name

EXISTS R

… IN R

… > ALL R

… > ANY R

NOT 🡪 works with all of these

LIKE keyword

* Search for strings without an exact match
  + Similar pattern in text

**%**

* Matches any sequence of 0 or more characters in string

**\_**

* Matches any single character in string

SELECT DISTINCT title FROM Movies

SELECT AVG(netWorth) as Average FROM Movie Exec

SELECT COUNT(DISTINCT title) as num FROM Movies

Deletion is different than dropping a table and alter table drop column 🡪 DELETE FROM R WHERE *condition*

DELETE FROM Movies

WHERE producerCert# IN

(SELECT cert# FROM MovieExec WHERE name = ‘John Smith’)

UPDATE Movies SET length=175, genre=’comedy’ WHERE title=’Star Wars’ AND year = 1977

**ER Diagrams** – Entity is an abstract object with attributes. An entity set is the permutations of all entities of certain types. Attributes are the properties of an entity. Relationships connect two or more entities. Relationships have names and there are two kinds of relationships: has a and is a. Is a relationships are like inheritance and polymorphism. **Rectangles** become an **entity set Attributes** are **ovals connected to a rectangle** with *no* **arrows.**

* Rectangles can be used when there are a lot of attributes to hold them all with the entity title

**Relationships** are depicted as **diamonds. Lines** connect **two entities** to the **relationship** to form an ER Diagram. A singleton relationship connects between two instances of the same entity. Multiway relationships are relations that connect many entities. Inheritance is represented by a triangle in the arrow to an entity. A weak entity set is an entity where its key consists of attributes from another entity. Is a relationships create an issue that we can resolve in three ways. The subclass can become its own table with each row having a key that identifies that base table that it is associated with (Not great). No base class table, just subclass tables with base attributes repeated (Dr frees hates this and it makes basic queries really difficult). One table for the entire hierarchy with nullable columns corresponding to all subtype properties (makes things easiest to query but can look confusing. Once we have a semi perfected diagram: Each regular entity will become a table and Each relationship will also become a table *if* It is a many-to-many relationship *or* if it had attributes attached to it that don’t make sense associated within one of the entities being related. Can make 1:1 and 1:N relations less complex: make the 1 to many an attribute of the many

* Sometimes you don’t want to do this though
  + If one attribute is complex, you might not want to condense