**Basic Terminology:**

Flat Files**:** Basic text file (BAD: inconsistent, doesn’t separate progr. From data)

File -IO:Reading/Writing to a flat-file **| Record:** One row in a flat file

DBMS adv: consistent, secure, more tools, multiple people can access data

DBMS disadv**:** flat files are free, DBMS requires staff, very complex systems

**DB Roles**:

1. Database Administrator (DBA)
   1. Give access to people, install/upgrade DB, monitor database
   2. Doesn’t *create* the DB, just run software using it
2. Database Designer (DBD)
   1. Collect/identify data to be used, identify relations b/w data
   2. use data to structure/model DB, talk w/ end users to know req
3. System Analysis & Application Programmer
   1. Don’t design DB, but write code to *communicate* with DB
4. End User
   1. Access DB, update/maintain and query the DB

**Schema Architecture**

Schema: describe database but NOT data (eg. “var type”, int, 3 tables, etc.)

Instance: occurrence of data item described in the schema (Homer Simpson)

State: data in DB at a moment in time (can change)

1.External Level: Other people accessing DB -> shows data ea. user interested in

2.Conceptual Level: How DB is structured (normally Db is stored in tables)

3.Internal Level: Describes data storage, ways to get the data

Logical Data Independence: change conceptual w/o changing external (**GOAL**)

Physical Data Independence: change physical w/o changing conceptual (**GOAL)**

**ER Terms:** Entity = “Employee” Attribute = “Employee #” Value = “343-242-22”

Composite Attribute: can be broken down eg. “John Doe” is First/Last Name

Simple Attribute: cannot be broken down eg. Age = “45”.

Single Valued vs Multivalued**:** Age vs CollegeDegrees (can have many degrees)

Candidate Key: if there are more than one primary key, each are Candid. Keys

**ER Diagrams** Multivalued Att: double circle Derived Attribute: dashed circle

Person(1,N) M -> Books -> Hotel Room(0,M) M | Artist(0,M)1 -> paints -> Painting(1,1)M

Woman(0,M) 1 🡪 gives birth to 🡪 Child(1,1) M (Child is total participation)

**Note: whenever you have subscript (1,M) or (1,1) it is total participation**

0/More:  1/More:  1/only1:  also 0/1

**Relational DB:** Hierarchical Data Model: parent-child relationship, data redundancy | Network Data Model: Records & Sets

**Superkey:** set of attributes (columns) that uniquely define a row in a table

**Candidate Key:** least # attributes which makes data unique

**Primary Key**: one key unique for all data (CANT BE NULL)

**Foreign Key:** key in one relation which is a primary key of another table

**Mapping ER to Relational Database**:

1.Start with regular entities | 2. Then map weak entities | 3. one to one

4. One to many | 5. Many to Many Rel. | 6. Multivalued relationships

**Mapping 1 to 1:** mapped normally, can add extra cols/rows

**Mapping 1 to Many:** move key from 1 side as column (NOT KEY) in many

**Mapping Partial/Total:** move key from partial to total (NOT KEY IN NEW)

**Mapping Many to Many:** Create a new table, both keys become new keys there

**Mapping Weak Entities**: Take key from owning entity and add to weak entity, make the weak attribute also a key

**Building** (BuildingID) (NumFloors) 🡪 has 🡪 **Room** (RoomNum) (floor) (doors)

* Now Room table has (doors, floor, RoomNum, BuildingID)

**Mapping Multivalued:** New table w/ multival and Primary Key from owner table

**# total tables:** Each entity gets 1, each M-M relationship is another, multivalued another as well. Then count them all.

**Physical Storage of DB**

Primary Storage:cache, main memory, very fast (no moving parts) use queries

Secondary Storage: Hard drives, flash memory, slower than primary

Tertiary Storage: Slowest, used for backups

Don’t use memory all time b/c cost, volatility (no backup), DB too large (size)

DB usually stored on hard drive online (good b/c online, bad b/c slow and $$)

DB also stored on flash memory (SSD): very fast but bad b/c no overwrite blocks

When designing DB, must decide on file organization + how to access other attr.)

**Magnetic Disk – The Hard Drive**

Track: Each disk broken down into track

Sector: can hold more data further it is but loser to centre = smaller # data storag

Seek Time: time to find the track Latency Time: time to find sector

Data on disk stored in form of records, file is sequence of records

**3 types of records:** 1. Fixed length 2. Two Var length fields + 3 fixed length fields

3.Variable field record w/ 3 fields **Block** = sector/group of sectors

**File Records can be:** spanned = records extend to another block (linked)

Unspanned: if record cut off, continue with next one

**Ways to add/edit/delete records on Hard Drive**

Heap Organization: add record to end of HD (fast insertion, very simple) Modifying/deleting expensive b/c searching linearly, **good** when data bulk loaded, when relation has indexed keys, **bad** for search & sorting & deletions

**Q: R=100,000 records w/ Block Size B = 2048 Bytes, each R = 500 Bytes**

Blocking Factor = 2048/500 = 4 Records per Block

# blocks needed = 100000/4 = 25000 Blocks

Linear Search = b/2 = 25000/2 = 12500 Block Accesses, worst avg case = b

Ordered Organization: add field to uniquely defined records and order them on that | Adv: searching fast, no sorting needed | Disadv: insert/delete/search=exp.

**IF YOU KNOW KEY, time is LOG2B, ELSE time is b/2 (same as heap)**

Hashing: 1 field is hash key K, blocks divided into M equal buckets K mod M

USE WHEN: get records using key field, want to distribute data evenly

**Q: Assume have records 34,44,22,24,23,100,46,50,32,61, there is 4 buckets, each bucket is 1024 bytes, each record is 333 bytes**

Each bucket can hold 3 records so **bucket 0** 🡪 1, 2, 3 **bucket 1** 🡪 1, 2, 3 etc.

If collision happens, open addressing = find next open slot, chaining = pointer to overflow area with empty buckets, multiple hashing = use 2nd hashing function

Hashing bad: no order, need fixed space, not good for att. other than hash key

**Indices**: Dense: index has entry for every search val Sparse: only some search val

Primary Index: ordered by ordered key field, 1 index entry per block record

Clustering Index: ordered on **nonkey** field so ordering may not be distinct

Secondary Index: block pointer points to block pointers (dense index)

**Q: 100,000 Records, block size 2048, each r = 500. Worst case Binary Search on Dense index?**

Each key field k = 10, block pointer p = 7, size of index record is 17bytes p/sec

Blocking factor for index file = 2048/17 = 120. Can hold 120 key fields and BP on 1 block

# blocks needed for index = 100000/120=834 blocks.

Log2834=10 block accesses + 1 (secondary file) = 11 block accesses

**Q: Same as above but what is search on Sparse Index?**

Block Factor = 2048/500 = 4 | # blocks needed = 10000/4=25000

Ordering Key Field same as above which is 17 bytes per second

BF for index = 2048/17 = 120 | Total # Entries = Total # blocks = 25000

# blocks needed for index = total # blocks / BF for index = 25k/120=209 blocks

Binary Search = log2b = log2(209) = 8 + 1 (index) = 9.

**Multilevel Index:** index on index | 1st level: original file 🡪 index 2nd = ind-index

* Uses a B+ tree where each level is index pointing to tree

**Relational Algebra**

Procedural Language: tell us how to get the data (Relational Algebra)

Non-Procedural Language: Tells us what data is needed (Relational Calculus)

**Selection:** return requested tuples (ROWS) as row subset

* σ Age > 30 (EMPLOYEE) *returns all rows where employee > 30*
* σ (ProjectLocation = ‘Toronto’ or ProjectLocation = ‘London’) (Project) returns all project info about projects located in London or Toronto

**Projection**: returns requested attributes (COLUMNS) (no dupl.)

* π LastName, FirstName (EMPLOYEE) *returns all employees (no dup)*
* π Address,FirstName( σ (Sex=‘M’) (Employee)) *returns address and first name of all male employees*

**Cartesian Project**: Concatenate 2 tables (every tuple mixed)

* creates new table from given 2 tables where every row in new table matches each row from each table

**Union:** Creates new table by merging the two and removing DUPLICATES

* Union compatible IFF have same # columns AND same type of vals (int)
* Temp1(Loc) 🡨 πProjectLocation (Project)
* Temp2(Loc) 🡨 πAddress (σ LastName = “Simpson” (Employee))
* Result 🡨 Temp1 U Temp2
* ^ “Show me all the project locations cities together with the cities that the Simpson’s employees live in”

**Full Outer Join:** R S: include all the columns from the first table and second table and put the equal values in the same rows, If not equal make rows NULL

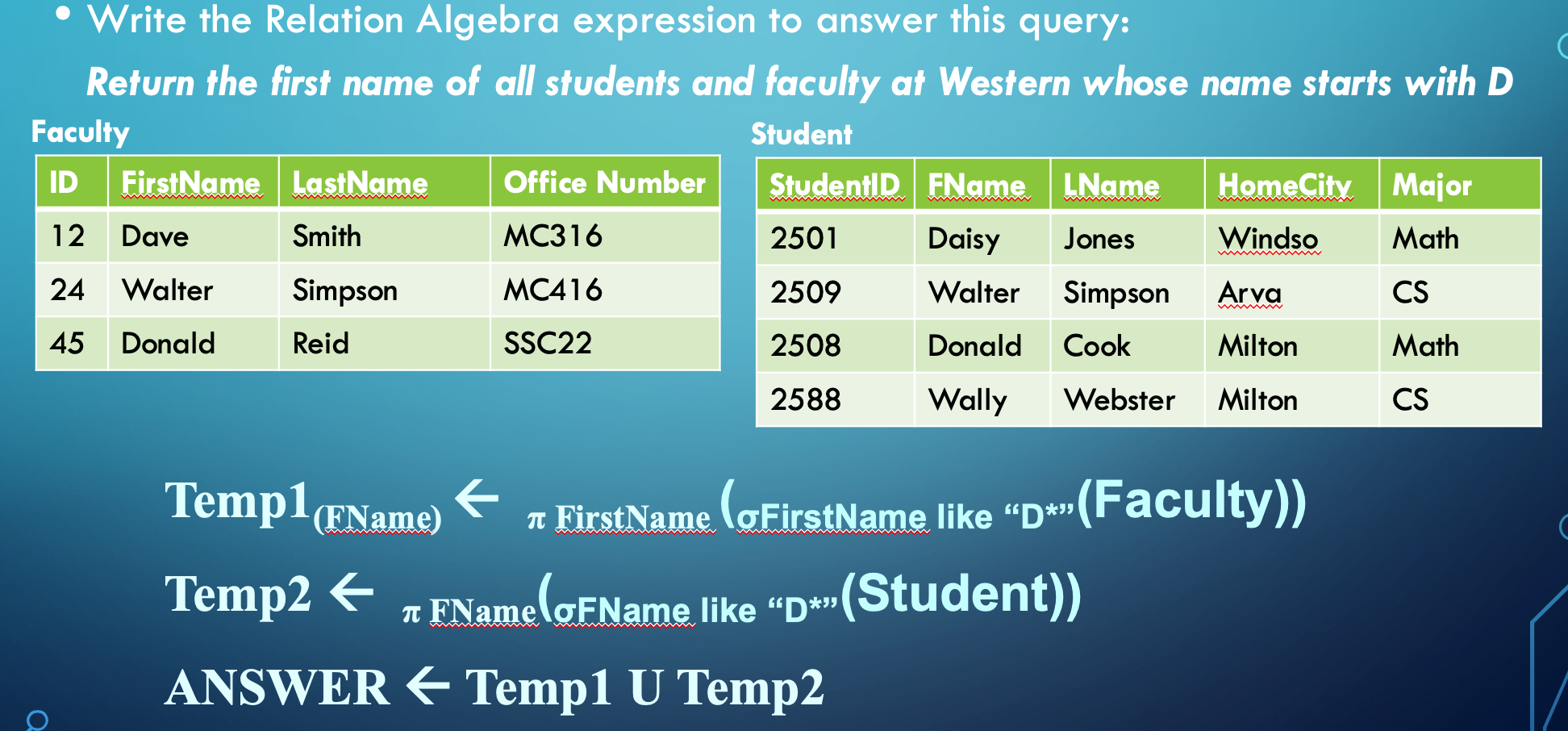
**Equi Join:** When all comparisons are =, pairs of attributes returned that are equal (attribute from both sides of equals are returned)



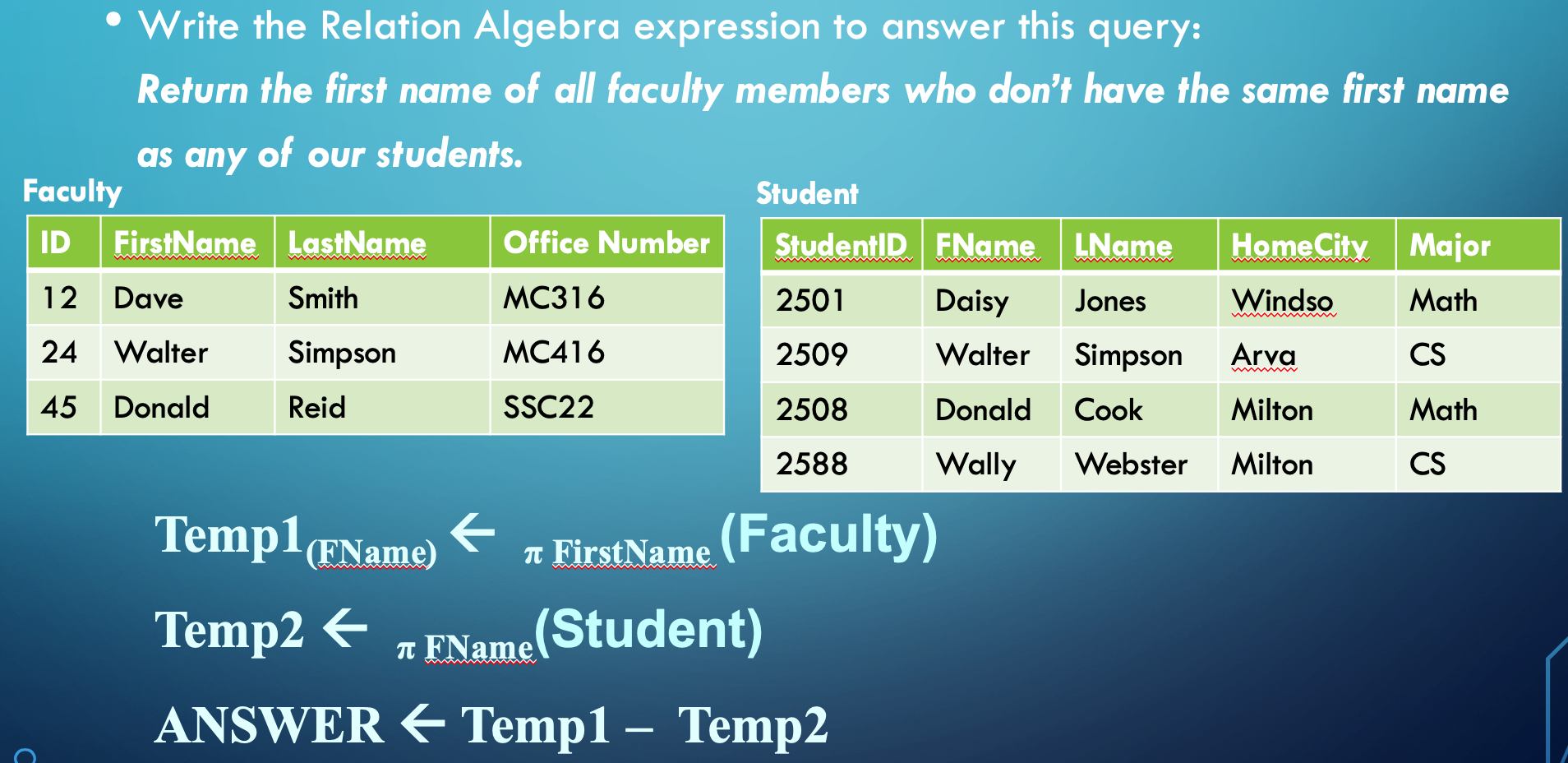
**Natural Join**: when all comparisons are = and it matches any attribute in table 1 which same name as attribute in table 2. Attribute only shown once.



Relational Algebra w/ Union, Projection/Selection



Relational Algebra with Set difference, projection/selection



**SQL**

INSERT INTO TableName VALUES (val1, val2, val3); (HAS TO BE IN SAME ORDER)

***OR CAN DO*:** INSERT INTO tableName (att1, att2, att3) VALUES (val1, val2, val3);

**Modify table:** UPDATE TableName SET AttrName = ‘Value’ WHERE Condition;

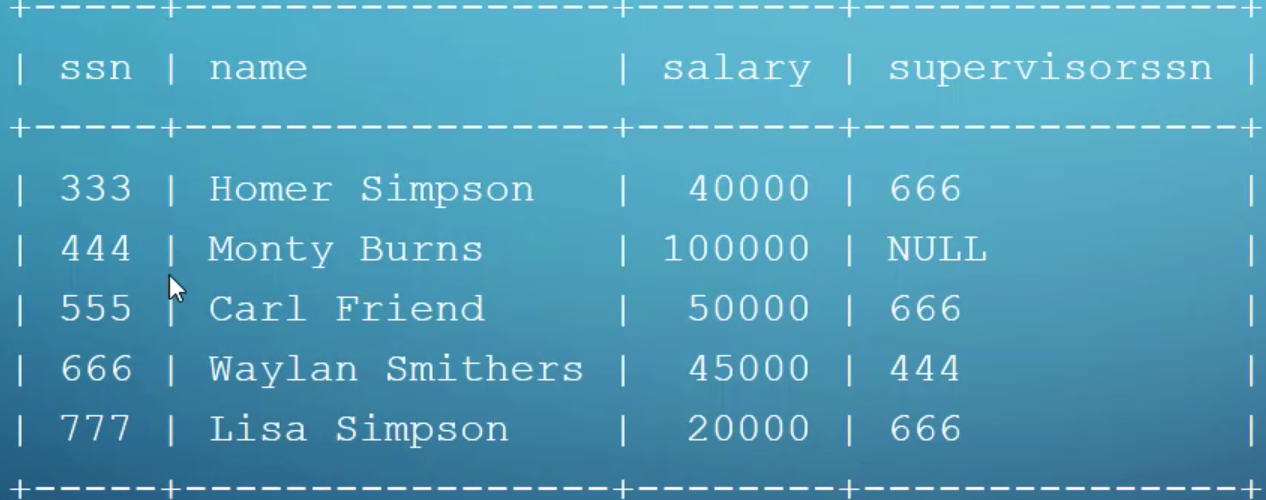
SELECT \* FROM TableName **OR** Select DISTINCT Name FROM employee (no dupl)

DELETE FROM owner; -- *deletes every single row of data unless there’s condition*

Describe table: shows you the columns and their values in the table

*\*If foreign key in the table to delete, can’t delete it b/c other table referencing it\**

**Views:** CREATE VIEW name AS SELECT FName,LName FROM table WHERE petID=22 |*Views don’t STORE data like table, but allows u to query “virtual table”*



*Q: Find all bosses and their workers: (have to use aliases: “name as emp Name”)*

A: SELECT e.name AS “Supervisor Name”, e2.name AS “Employee Name” FROM emp e, emp e2 WHERE e.ssn=e2.supervisorssn; (join on itself using temp table)

Q: Find employees who work in London-located departments

A: SELECT \* FROM Employee WHERE DeptNo IN (SELECT DeptNo FROM Department WHERE deptlocation = “London”)

Q: Find last name of people who don’t drive any of the cars in the DB

A: SELECT LastName FROM Driver WHERE DriverLic NOT IN (SELECT DriverLic FROM Car)

**Triggers:** set of SQL statements that execute when certain events occur in table

CREATE TRIGGER upd\_check BEFORE UPDATE ON account

BEFORE = run trigger before user does update AFTER = do trigger after

UPDATE = if user updates table INSERT = when val inserted DELETE = val deleted

\*Can’t have 2 of the same triggers for a table (eg. Both are BEFORE INSERT)

**Stored Procedure**: procompiled application written in any lang executed in reponse to a SINGLE SQL statement (makes SQL logic easier to execute)

Adv: faster b/c code is compiled, reduces traffic, reusable and secure

Disadv: too many proc can slow down, hard debug, migrating DB tricky

**System Catalog:** group of tables that describes all our databases

Can’t ever write inserts/deletes/updates for sys tables, but can do select

Information\_Schema: holds all tables we created [all automatically]

Performance Schema: tells us how fast queries are created

**Database Security**: 2 mechanisms: Discretionary and Mandatory

Discretionary: give privileges to users on records, fields, etc. (like chmod)

Mandatory: multiple security levels and categorize users on those levels

Account Level: access to DB Relation Level: access to data

Security Levels: 1. Top Secret 🡪 2. Secret 🡪 3. confidential 🡪 4. Unclassified

User can only read if their class is greater than or equal to object reading from

User can only write if their class is less than or equal to object writing to

Statistical DB Security: user’s can’t select data 🡪 don’t want low result returns

Spurious Tables: extra rows/tuples | Anomaly: inconsistencies with data manip.

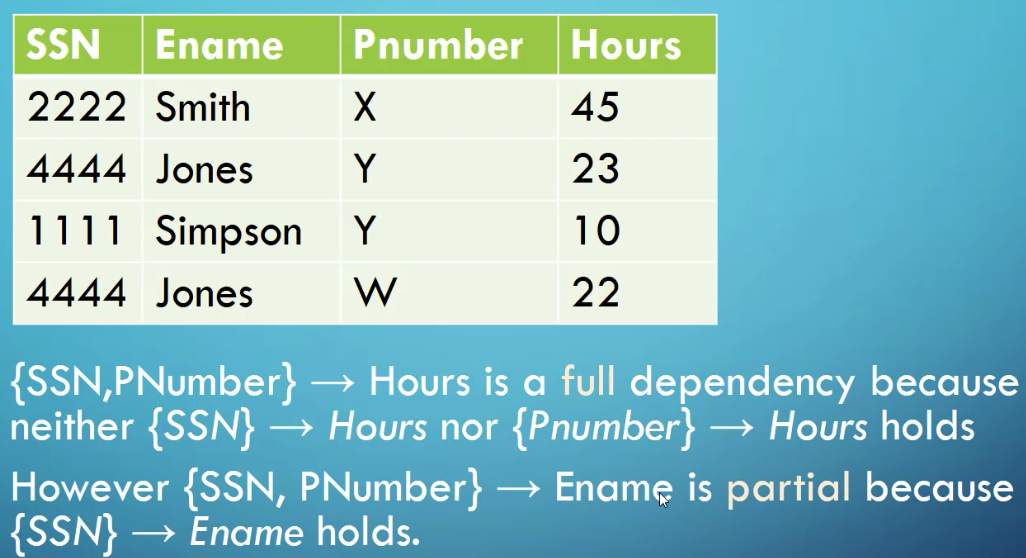
**Normalization**: assigning attrib. to correct tables to reduce repeats/inconsistency

**First Normal Form**: every row should have exactly 1 value

**Functional Dependency:** if given an attribute, can I get value of another attrib?

*Ex. Given EmpID can get firstname* ***&*** *given lastname cant get grade (repeats)*

**FD** cannot be inferred automatically from DB (must ask someone from company)

**FD x 🡪 y is full FD if removal of attribute A from X means dependency ≠ hold**

**FD x 🡪 y is partial FD if removal of attribute a from X still makes it dependent**

**Second Normal Form:** all of the attributes should be FD on all primary keys

*To put it into 2ndNF, split them up so they’re in separate tables that depend on each other*

**3rd Normal Form:** if in 2nd NF but one attribute determines another attribute (Ex. Sin# **determines** Name, Dept ID, Dept Name, but Dept ID determines Dept Name

**Lossless Join Property**: when split table into subtables