WEEK ONE:
Data: Known facts that can be recorded and have an implicit meaning.
Database: A collection of related data.
Mini-world = Data we extract from (ex: Western)
Database Management System (DBMS): A software package/system to facilitate the creation and maintenance of a computerized database.
Database System: The DBMS software together with the data itself. Sometimes, the applications are also included. **Flat File:** .txt or. askii file, they are inconsistent
Record: Each row of data or "instance"
Advantages of DBMS over flat file system: Data redundancy/inconsistency reduced, Isolates the programs from the data, and multiple users can use and share data, more abstract conceptually(views)
Scheme: A description of the database but NOT the data itself, ex: Patient(patient, name, OHIP Number)
Instance: An occurrence of a data item described in the schema – ex: The patient name might be homer simpson and the ohip number might 123-345-678
State/Snapshot: The data in the database at a moment in time, anytime we delete or add a value of a data, we change the state
3-Schema Architecture: Internal Level: Describes the daya stograge to get data, **Conceptual Level:** Data is stored in tables, **External Level:** Each view only shows the data that each end user is interested in, and hides the other data
Data Independence: Logical Data Independence: Capacity to change the conceptual schema without affecting the external schema or views. **Physical Data Independence:** Ability to change the physical schema without changing the conceptual schema.
Entity Relationship Model: Created by Peter Chen 1976.
Entities and Attributes: Entity - A single "THING" that exists, has independent existence ex: Employee
Attribute – describes a "thing" ex: Age, SSN, Sex, Name
Value - taken on by an attribute ex: 25, 456-876-788, Female, Bart Simpson **Composite Attributes vs. Atomic or Simple Attributes** ex: Bart Simpson vs. **45 Single-Valued Attributes vs. Multivalued Attributes** ex: Age vs. College Degrees **Derived Attributes vs. Stored Attributes** ex: Age vs. Birthdate
ENTITY any object in our mini-world that we want to model and store information about. E.g. Student, Professor, Classroom
ATTRIBUTE defines the information about the entity that needs to be stored. An entity will have 0 or more attributes. An attribute has a DOMAIN. E.g. student number
DOMAIN the type of values that an attribute can take. E.g. String, integer, real, date
ER Diagram Notations: Entity = rectangle, attribute = oval, key attribute = oval, but w/ underline, multivalued attribute = double oval, and composite attribute = multiples ovals connecting to it, and derived attribute = oval in dashed lines. We used a DIAMOND to represent a relationship.
Recursive (Unary) Relationship: an entity of one entity type has a relationship with other entities of that same entity **Attributes on Relationships:** Describes some piece of information about the relationship. E.g. Quantity
Cardinality Ratio: number of relationships instances that an entity can participate in, there are 3 common ones: **One-To-One:** Employee Manages **Department Many-To-One**: Employee Works_For Department **Many-To-Many:** Employee Works_On Project
Participation Constraint: specifies whether the existence of an entity depends on it being related to another entity via the relationship type
Total (Mandatory) - every entity in the entity set MUST BE related to the other entity set via the relationship. (For example, every employee must Work_For a department) SHOW ON ER DIAGRAM WITH A DOUBLE LINE
Partial (Optional) - some or part of the entity set are related to the other entity set but not necessarily all. (For example, some employees manage a department but not all) SHOW ON ER DIAGRAM WITH A SINGLE LINE
Other notation --> (min, max) where 0<=min<=max and max>=1. Each entity must participate in at least min and at most max relationships. Thus a min of 0 implies partial participation. Partial = (0,M) and Total = (1,M)
Weak Entity: Can't exist without owner, have no key attribute of their own. Double outline the relationship and weak entity
WEEK TWO:
In **Hierarchial models**, only 1:n representations can be modelled. It has problems modelling M:N relationships, the case where a record type participates as child in more than 1 PCR type, N-ary relationships with more than 2 participating types.

Network Data Model: Designed and created by the **CODASYL**. **Two basic data structures:** Records and Sets. Record ex is: student, class, course.
Key(s) → combination of attributes (or a single attribute) that can be used to enforce that no 2 tuples can be identical
SuperKey: any set of attributes that enforce that no tuples are alike
Candidate Key: sometimes a table will have 2 possible things that could be keys (e.g. employee number and SSN, each of them is a candidate key)
Also must be a superkey such that proper subset is a superkey within the relation. Candidate key is a "minimal superkey"
Primary Key: Pick one candidate key to identify tuples in the relation (signify this key by underling it), could turn out to be composite key (2 or more attributes combined together)
Foreign Key: An attribute or set of attributes within one relation that matches the candidate key of some other (possibly the same) relation
Key Constraints: Keys must be unique, primary key must be non-null, most DBMS enforce both of the above constraints
A Domain D is a set of atomic values, Example: USA_Phone_Numbers, Employee_Ages, Department IDs **ex:** Set of 10 digit phone numbers valid in Canada
Possible ages of employees A data type is specified for each domain like 10 char string or positive integer
A Relation Schema R denoted by R(A1, A2, ...An) is made up of a relation name and a list of attributes. Each attribute Aj is the name of a role played by some domain D in the relation schema R.
A Relation (or relation state) r of a relation schema R(A1, A2, ... An) also denoted by r(R) is a set of n-tuples r = {t1, t2, ... tm}. Each n-tuple t is an order list of n values t=< v1, v2, ... _____> , where each value vi, 1<=i<=n, is an element of dom(Ai) or a special null value.
R is called the name of the relation schema
Attribute is a named column in a relation schema
Tuple is a row of a relation
Degree of a relation is the number of attributes it contains
Cardinality of a relation is the number of tuples it contains
Properties of Relations:
Each relation name is unique
Each cell in a relation contains 1 atomic value Normalized, First Normal Form
Each attribute name within a table is unique
The values of an attribute are from the same domain
The order of the attributes has no significance
Each tuple is distinct (no duplicates)
The order of the tuples has no significance (Tuples in a relation do not have any particular order, however in a file records are physically stored on disk so there is always an order among records. Note: we may chose to display the records in a particular order)
***Relational Databases** can only use these data structures:*
Tables (Relations) Rows Columns Cells.
REPRESENTING 1 to Many Relationships in Relational Database Model: The rules are: 1. YOU CAN ADD AS MANY NEW COLUMNS AND ROWS AS YOU WANT TO THE EXISTING TABLES 2. AND IF YOU NEED A NEW TABLE YOU CAN ADD THAT ALSO, 3. BUT THAT IS ALL YOU CAN ADD COLUMNS, ROWS and TABLES
In Many To Many relationships, you make a NEW table and the key for the new table is the combination of the keys from the entities participation in the many to many relationship. Also include any attributes on the relationship in the new table.
With WEAK entities, you bring the key from the owning entity as part of the key for the weak entity, so make the owning key a new column in the weak entity table and combine it with the weak key to make the new key.
With MULTIVALUED attributes, you create a new table and bring the key from the entity as part of the key together with the multivalued attribute and combine them to create the new key for the new table. Do NOT include the multivalued attribute in the original entity anymore.
.
Three Main Types of Constraints: Key constraints Referential integrity constraints Semantic integrity constraints
Key Constraints: **PRIMARY KEY** - Allows you to state which attribute(s) will be the primary key (the attribute(s) that ensures that no 2 tuples are identical). A table can only have ONE primary key but the primary key can be made up of several attributes The primary key MUST be unique **NOT NULL** – Forces the user to never leave a key attribute null (empty) for a particular tuple.

Referential Integrity: a tuple in one relation (table) that refers to another relation (table) must refer to an existing tuple in the relation.
Semantic Integrity Constraints: **State Constraints:** state the constraints that a valid state of the database must satisfy Example: Hours worked cannot be greater than 50, Quantity Ordered must be greater than 10 **Transition Constraints:** define how the state of the database can change Example: Salaries can only increase Both of the above are enforce in relational databases through **triggers and assertions**
WEEK THREE:
Storing Database in Computer: **Primary Storage** CPU (Main Memory and faster Cache memory) **Secondary Storage** Magnetic Disks Flash Memory (Solid State Drive SSD) **Tertiary Storage** Optical Disk (CD-ROM, DVDs) (declining because of decrease in cost and increase in capacity of magnetic disks) Tapes
Hard drives are slower than main memory because of **Seek time** – find the track (move the arm to the correct circle) **Latency** (rotational delay) – find the sector (spin the disk to the correct pie area) **Block Transfer Time** – move the data (Seek and Latency take much more time)
Placing Data on the Disk: Data is stored in form of records: fixed length or variable length. **Collection** of related data values where each value is a byte. **File** is a sequence of record.
Blocks: Sequence of bytes, a whole block can be brought into main memory, thus the size of a block is related to the OS
QUESTION: Find the average search time to find a record if you use a heap organization for the following scenario: r = 100,000 records stored on a disk with block size B = 2048 bytes. Each record (R) is a fixed size of R = 500 bytes. Blocking Factor (bfr) = 2048/500 = 4 records per block # of blocks needed is 100k/4 = 25k blocks
Linear Search: on average b/2 = 25k/ 2 = 12500 block accesses
Binary Search: Worst Case: log2b = log2(25000) = 14.6(15) block accesses
Advantages: Searching on the key field can use a binary search No sorting required **Disadvantages** Inserts and Delete can be expensive Binary search on a disk can be very expensive
Open addressing: Find the first open position following the position that is full **Chaining**: Overflow area is kept and a pointer to the overflow area that is use **Multiple Hashing:** another hash function is applied to the record if the first results in a collision
Index: Dense index has an index entry for every search key value **Sparse (non-dense) index** has index entry for only some of the search values
Primary Index: Always defined on an ordered data file Ordered on the ordering key field (but might not be the primary key of the table) Includes one index entry for EACH BLOCK in the data file (not for each record)
Clustering Index: Defined on an ordered data file Ordered on a non key field, thus every ordering field might not be distinct Includes one index entry for each distinct value in the ordering field Insertions and deletions are straightforward if each cluster starts a new block
Secondary Index: Provides a secondary access for a file which already has primary access Might be on a candidate key or any field Can have many secondary indexes Includes one entry for each record, hence it is DENSE
Rules for B+- TREES: If the root is not a leaf node, it must have at least 2 children For a tree of order n, each node (except root and leaf) must have between n/2 and n pointers and children. If n/2 is not an integer, round the result set up For a tree of order n, the number of key values in a leaf node must be between (n-1)/2 and (n-1) pointers and children. If (n-1)/2 is not an integer, round up the result The number of key values contained in a non-leaf node is 1 less than the number of pointers The tree must always be balanced: i.e. every path from the root node to a leaf must have the same length Leaf nodes are linked in order of key values
WEEK FOUR:
Procedural relational languages, they tell us how to get the data (e.g. Relational Algebra) **Non--Procedural,** they tell us what data is needed (e.g. Relational Calculus)
Five Basic Operations in Relational Algebra:
Selection: Unary (works on 1 Relation (TABLE) only), returns only the tuples from a relation that satisfy the specified condition. (i.e. returns a row subset), written as: σ condition (R) **Projection**: Unary (works on 1 Relation only), returns only the requested attributes (with no duplicates) (returns a column subset), written as: π attribute1, attribute2 (R) **Cartesian Product:** Binary (requires 2 Relations), returns a relation that is the concatenation of every tuple of relation R with every tuple of relation S, Symbol: S X R **Union**: Binary (requires 2 Relations), union of relations R and S with I and J tuples, respectively, is the concatenation of them into one relation with a maximum of I+J tuples, duplicate tuples being eliminated, R and S must be union compatible (i.e. R and S must have the same columns or attribute domains). Symbol: R U S **Set Difference**: (requires 2 Relations), R-S = a relation consisting of the tuples that are in relation R but not in S, R and S must be union compatible. S – R
Selection: Example Expression: σ ← symbol for selection Age > 30 ← condition that each row must satisfy to be returned in the answer (EMPLOYEE) ← Table name
Projection: π ← symbol for selection Age,LastName ← the columns (atrebutes) that should be returned (EMPLOYEE) ← table name
Union: Two tables are union compatible, if and only if: They have the same number of columns Each respective column from each table is from the same domain.
Creates a new table from the given 2 tables that include every row from both tables with NO repeating identical rows. The 2 Tables MUST be union compatible Symbol U **Example Expression**: Table1 ← first table name U ← symbol for union Table2 ← second table name
Difference: Create a new table from the given 2 tables that include every row from the table on the left side that is NOT in the table on the right side. The 2 Tables MUST be union compatible Symbol '–' **Example Expression:** Table1 ← table 1 – ← symbol for difference Table2 ← table 2
Cartesian Product: Creates a new table from the given 2 tables where every row in the new table is a match of each row from each table. The new table will have all the attributes of the first table AND all the attributes of the second table The new table's number of rows will equal first table's number of rows * the second table's number of rows. **Symbol X Example Expression**: Table1 X ← Table2
Join: A join is just a Cartesian Product X with a Selection σ to find matches. The selection will remove some of the rows/tuples from the returned Cartesian Product. The attributes that will be in the new table depends on if you are doing an equi join or a natural join Symbol ⋈ Example Expression: Table1 ⋈ Table2 (natural join) Table1 ⋈ columnnametable1=columnnametable2 Table2 (equi join)
Equi Join: when all of the comparisons are =, then it is called an equi join and pairs of the attributes are returned that are equal (i.e. attribute from both sides of the equals are returned). **Natural Join**: when all of the comparisons are = and it matches any attribute in Table1 that has the same name as the attribute in Table2. The attribute is only shown once.
Full Outer Join: A full outer join is similar to a join except that it includes all the rows from both tables even if they don't have a matching value in the column that you are joining. If there is no match, put nulls in the columns from the other table. Table1]X[Table2 (outer natural join) Table1]
X[columnnametable1=columnnametable2Table2 (outer equi join)
Full Outer Join: R]X[S: a join in which tuples from R that do not have matching (equal) values in the common columns of S still appear and tuples in S that do not have matching values in the common columns of R still appear in the resulting relation (padding the fields with nulls)
Left Outer Join: R]X[S : a join in which tuples from R that do not have matching values in the common columns of S still appear in the resulting relation
Right Outer Join: R]X[S: tuples in S that do not have matching values in the common columns of R still appear in the resulting relation.
Intersection: Creates a new table from the given 2 tables that includes only the identical rows from both tables (no repeats). The 2 Tables MUST be union compatible Intersection can be expressed as: (R U S) - ((R - S) U (S - R)) Symbol → ∩ **Example Expression**: Table1 ∩ Table2
useful in situation with the word *both* or *and*, such as list the people who work on BOTH project X and project Y ProjectX ← π ProjectNumber (σ ProjectName = "X" (Project))
Division: Useful for situations where the term "ALL" is used, for **example**: Find the first and last names of employees who work on all the projects that Dave Leno works on. the answer would be like this:
Leno ← π EmpID (σ LastName = "Leno" (Employee))
Leno_Proj ← π ProjectNumber (Works_On ⋈ EmpIDSSNum = EmpID Leno)
All_Proj ← π EmpIDSSNum, ProjectNumber(Works_On)
LenoPROJ ← All_Proj ← Leno_Proj
Result ← π FirstName,LastName (LenoPROJ ⋈ Employee)

Table 6.1
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\langle \text{join condition} \rangle} R_2$, OR $R_1 *_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

WEEK FIVE:
MySQL COMMANDS: SHOW lists all databases DROP deletes a database **CREATE** creates a database **USE** allows you to start using a database
EXAMPLE: CREATE TABLE owner (firstname VARCHAR(20), lastname VARCHAR(20), ownerid INT NOT NULL, PRIMARY KEY (ownerid));
DROP TABLE TableName;
INSERT: INSERT INTO TableName (Attr1Name, Attr2Name, Attr3Name) VALUES (value1, value2, value3);
EXAMPLE: INSERT INTO owner (firstname, lastname, ownerid) VALUES ("Laura","Reid",1);
Update Existing Data: UPDATE TableName SET AttrName = 'Value' WHERE Condition;
Example: *UPDATE owner SET firstname="Elaine" WHERE firstname ="Laura";*
Delete tuples from tables: UPDATE TableName SET AttrName = 'Value' WHERE Condition;
Example: *UPDATE owner SET firstname="Elaine" WHERE firstname ="Laura";*
Basic Queries: SELECT AttrName[, AttrName] FROM TableName[,TableName] WHERE Condition;
Example: *SELECT name FROM pet WHERE species ="Dog";* (OR) *SELECT * FROM pet, owner WHERE pet.ownerid=owner.ownerid;*
(this is a join)
Creating Views: CREATE VIEW ViewName AS SELECT AttrName[, AttrName] FROM TableName[,TableName] WHERE Condition;
Example: *CREATE VIEW ownersandpets AS SELECT firstname, lastname, name FROM pet, owner WHERE owner.ownerid=pet.ownerid;*
*SELECT * FROM ownersandpets;*
Why do we need Views? A searchable object that doesn't STORE data like a table but allows you to query it like a table, a virtual table (retrieves data from the real tables that are being combined together)
Examples for Everything:
Query: Find the employee who makes the greatest salary.
Example: mysql> SELECT name,salary FROM emp WHERE salary = (SELECT MAX(salary) FROM emp);
Query: Find all departments which have projects in a city that is not the same location of the department.
*SELECT * FROM Department AS d WHERE EXISTS (SELECT * FROM Project WHERE d.location <> projectlocation AND d.deptno = Project.deptno)*
WEEK SIX:
Two types of relational calculus, Tuple Relational Calculus Domain Relational Calculus
General Form: {t | COND(t)}
where t is a tuple variable and COND(t) is a Boolean expression involving t **COND(t)** must evaluate to be TRUE for t in order to return that tuple. **Example**: To find all employees whose age is > 45, we write the following: {t | EMPLOYEE(t) AND t.Age > 45} **if** we only want first

and last name for employees over 45 we write: {t.Lname, t.Fname | EMPLOYEE (t) and t.Age > 45}
QUESTION: Retrieve the birth date and address of the employee whose name is 'Jon Mortensen' assuming this is one of your tables (relations): ANSWER:{t.Bdate, t.Address | EMPLOYEE(t) AND t.Fname="Jon" and t.Lname="Mortensen"}
Also we need two more symbols called quantifiers: they are the universal quantifier → ∀ and the existential qualifier → ∃
With the existential quantifier, a formula:
(∃t)(F) is TRUE if the formula F evaluates to TRUE for some (at least one) tuple. If you are NOT displaying a tuples attribute in your answer, but you need that tuple from another table to do the join, then you MUST use (∃t)(F)
With the universal quantifier, a formula: (∀t)(F)
is TRUE only if the formula F evaluates to TRUE for every tuple.
Ex: Retrieve the name and address of all employees who work for the *'Research'* Department
{t.Fname, t.Lname, t.Address | EMPLOYEE(t) and ((∃d)(DEPARTMENT(d) and d.Dname = 'Research' and d.Dnumber = t.DNO))}
Ex: For every project located in *'London'*, list the project number, the controlling department number, and the department manager's last name.
{p.PNumber, p.Dnum, e.Lname | PROJECT(p) and EMPLOYEE(e) and p.PLocation = 'London' and ((∃ d)(DEPARTMENT(d) and d.Dnumber = p.Dnum and d.Mgrssn = e.ssn))}
Domain Relational Calculus: In Domain Calculus an expression is of the form: {x1, x2, ..., xn| COND(x1, x2, ...xn, xn +1, ...xn+m)} **Retrieve** the birth date and address of the employee whose name is 'Jon R. Mortensen' : {u,v | (∃q)(∃r)(∃s) (EMPLOYEE(qrstuvwxyz) and q = 'Jon' and r = 'R.' and s='Mortensen')} or
alternative notation would be: {u,v | EMPLOYEE('Jon','R.','Mortensen', t,u,v,w,x,y,z)}
Specialization: Process of finding a set of subclass (subtypes) of an entity.
Each subclass will have some distinguishing characteristics of the superclass **Example**: laptop and desktop are subclasses of computer Remember the ISA rule → Only use this if you can say the subclass IS A superclass.**Example**: Laptop IS A Computer – YES (so laptop can be a subclass of Computer) Country IS A Continent – NO (so Country should NOT be a subclass of Continent)
The U on the line from Secretary to the d node indicates that the set of all instances of Secretary is a **subset** of the set of instances of Employee.
The *d* in the node indicates that these three subsets (Secretary, Technician and Engineer) are **disjoint**
Generalization: **The** opposite of Specialization is *Generalization* **Identify** common features and generalized them into a single superclass. **Sometimes** the instances of the subclasses overlap – indicated with an **o** in the circle.
WEEK EIGHT:
Triggers and Stored Procedures: Triggers are a set of SQL statements that execute when a certain event occurs in a table **Similar** to a constraint but more powerful: **Example**: a constraint will disallow a salary to go above a certain amount **A trigger** can check how much the salary changes and if the change is above a certain amount, it can cause a user-defined function to notify an administrator about the change. **Almost** all database management systems have triggers **Can help** with the following: If a business rule changes, you just need to change trigger not code Improved performance (rules run on the server) Global enforcement of business rules
QUESTION: What will cause the trigger to be activated? When you no longer need the trigger, do the following command:
DROP TRIGGER [IF EXISTS] trigger_name
QUESTION: What do you think happens if have a table called *account* and it has a trigger associated with it and you do the command: *DROP TABLE account?* All tables associated with that table will be dropped.
SITUATION: Suppose you need to update the salary of all 100,000 employees in your database, by 6.25%. You are working in Sydney, Australia with a java application on your desktop, the database is in New York City. What problems can you foresee?
QUESTION: What do you do to solve this problem?
ANSWER: Use a Stored Procedure!
Advantages of Stored Procedures:
Typically, faster, because the code for the stored procedure is compiled
Reduce traffic between application and database because it only has to send the name of procedure, not long and multiple SQL statements
Reusable
Secure – DBA can grant permissions to them without giving permissions to the underlying tables.
Disadvantages of Stored Procedures:
Over usage (using MANY stored procedures) can actually slow down the application (increased load on database server). Sometimes doesn't allow for complex business logic. Hard to debug. Migrating to another Database System can be tricky.
QUESTION: How do you think MySql represents your database?
MySQL represents your databases as a bunch of tables, just like you represent your database!
These tables are called System Tables or System Catalog or Data Dictionary or Metadata or a mini database which describes your database.
System Catalog keeps track of all the table names, attribute names, attribute domains, descriptions of constraints, etc...
QUESTION: so what does the SCHEMATA table hold?
ANSWER: all our databases (a row for each database)
QUESTION: What is the SCHEMA_NAME?
ANSWER: the UNIQUE name of each database (the primary key for this table!)
Question: how would you only show the table names in a particular database?
select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA = 'petoffice';
Write a query to list all the tables without a primary key
SELECT t.table_name FROM tables t LEFT JOIN TABLE_CONSTRAINTS tc ON t.table_schema = tc.table_schema AND t.table_name = tc.table_name AND constraint_type='PRIMARY KEY' WHERE tc.constraint_name IS NULL AND t.table_type='BASE TABLE';
Find the top 5 largest tables in a database
SELECT table_schema, table_name, data_length FROM tables ORDER BY data_length DESC LIMIT 5;
QUESTION: What do you do if you create a table and can't remember what type a particular column is?
ANSWER: SHOW TABLES; DESCRIBE TABLE;
QUESTION: What are some of the *things* (components) in a relational database: Databases, Tables, Columns, DataTypes, Views;
QUESTION: What is another word for a *thing* in an ER diagram? Answer = entity
Database Security:
Legal/Ethical Issues (right to privacy of information, ...)
Policy Issues (what does your company make available; who within the company should be updating what)
System Issues (where and how should security be handled? Physical hardware level, operating system level, DBMS level)
2 Types of Database Security Mechanisms:
Discretionary: grant privileges to users on tables, records, fields, etc. based on the discretion of the owner of the table (what you're used to in Unix)
Mandatory: Multiple security levels, categorize the data and the users on the levels; decisions of who gets to see what are based only on relative levels/security labels (example: Top Secret, Secret, Confidential, Unclassified)
Database Security and the DBA:
DBA has a system account (like a superuser). The DBA account performs the following actions:
→ **Account** Creation (sets account name, password)
→**Privilege** Granting →**Privilege** Revocation
→**Security** Level Assignments (set user account to appropriate security classification)
User must log on to DBMS using an account name and password **All** actions by the user are monitored in a log file **If anything** suspicious occurs a database audit is performed to check actions and accounts
Discretionary Access Control: **Grant** and revoke privileges from users, 2 levels
Account Level: giving the user access to the database in general →
CREATE SCHEMA ; CREATE TABLE ; CREATE VIEW ; ALTER, DELETE, MODIFY, SELECT
Relation Level: giving the user access to each relation in the database
→**Each** relation R is assigned an owner account (usually the account that created the relation in the first place). The owner account is given all privileges on that relation. This owner can pass on privileges to other users by granting privileges to their accounts
→**SELECT** privilege on R → Account can only retrieve from R
→**MODIFY** privilege on R (divided into UPDATE, DELETE and INSERT privileges). With the Insert and Update you can restrict to just certain attributes
→**References** privilege on R → Used for creating integrity constraints
Creating Users:
OR (for a user with no password) → WEAKEST
CREATE USER 'jeffrey'@'localhost';
OR → WEAK
CREATE USER 'lreid'@'localhost' identified by 'mypass';
OR → STRONGEST
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY PASSWORD
'*90E462C37378CED12064BB3388827D2BA3A9B689';
(This one is because if you do the first one, the actual password will go into the log file and it can be viewed)
To remove a user that you have created use the DROP command: DROP USER 'jeffery'@'localhost';
DB Privileges SQL Commands:
REVOKE → removes privileges
GRANT → adds privileges
WITH GRANT OPTION → allows one user to give another user the ability to give other users privileges
Suppose we only want *sskinner* to see the birthdate and name of employees but not salaries or other info for the Payroll department (then we would do the following:)
CREATE VIEW vlimitemp as SELECT name, bdate FROM employee WHERE DeptNo = '5';
QUESTION: Then what would we do? **Answer**: *GRANT SELECT ON vlimitemp TO sskinner;*
QUESTION: What do you think this statement does:
GRANT UPDATE ON Employee(Salary) TO bgumbel;
***ANSWER**: bgumbel can only update the salaries but no other fields!*
Three Types of Control:
→Role Based Access →Mandatory Access
→Statistical Access
Role-Based Access Control:
Used to design access control for complex systems:
→**Define** a number of roles, where each role is a set of privileges/permissions like {(insert, TableA), (delete, TableB)} →**Grant** roles to users. This saves a lot of time over granting of individual permissions.
Mandatory Access Control:
Typical Security Classes:
Top Secret (TS) ← highest Secret. (S) Confidential (C) Unclassified (U) ← lowest
Bell-LaPadula model classifies:
→**Subjects** (Users, Accounts, Programs), called the clearance, refered to as CLASS(S)
→**Objects** (Relations, Tuples, Columns, Views, Operations), called the classification, refered to as CLASS(O)
2 Rules are: 1)A subject S cannot read from an object O unless class(S) >= class(O) **2)A subject** S cannot write to an object O unless class(S) <= class(O)
QUESTION: Do you have any ideas/suggestions for how you could handle such leaks of information?
Limit queries if the result returns a value less than a certain threshold
Limit repeated queries that refer to the same tuples
Introduce "noise" (inaccuracies) into results to make it difficult to deduce individual information
WEEK NINE:
Normalization:
Normalization is the process of assigning attributes to entities
Normalization reduces redundancies and produces controlled redundancy that lets us link our tables
Some terminology: If a relation has more than one minimal key, each is called a Candidate Key and the one that is chosen as the key for the relation is called the Primary key and the others are called Secondary Keys
An attribute of R is a prime attribute of R if it is a member of any key of R, otherwise it is a nonprime attribute.
In the ProjEmp (Works_On) table, SSN and Pnumber are prime attributes and all the rest (Hours) are nonprime.
Normalization works through a series of stages, you move from first normal form to second normal form to third normal form
Normalization is not always desirable because generally the higher the normal form, the more pointer movements (or joins) are required to get output, thus sometimes after normalizing the database you may then denormalize some portions for performance reasons.
First Normal Form:
All the relations we have been building so far have already been in first normal form. **It disallows** multi-valued attributes, composite attributes and their combinations **To** be in first normal form: the domains and attributes must include only atomic values and the values of any attribute in a tuple must be a single value from the domain of that attribute
To put it into first normal form we have 3 options:
→Split out the repeating values and using the repeating value as part of the key. →Break it into 2 tablers→If you have a small LIMITED number of values that are repeated, you could make new attributes for those values

Functional Dependency:
Functional Dependencies is the main tool for measuring the appropriateness of your attribute grouping into relations.
Used to normalize into second and third normal forms.
X → Y (Y is functionally dependent (FD) on X) where X and Y are sets of attributes from relation instance r of R states that for any 2 tuples t1 and t2 in r such that if t1[X] = t2[X] then t1[Y] must = t2[Y]. I.E. the values of the Y component of a tuple in r depend on, or are uniquely determined by the X component (X functionally determines Y).
A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore.
A functional dependency X → Y is a partial dependency if some attribute A contained in X can be removed from X and the dependency still holds.
Inference Rules:
Reflexive rule (not really useful): if X is a set of attributes and X ⊇ Y, then X → Y *(or: If A is a set of attributes, and B is a set of attributes that are completely contained in A, the A implies B.)*
X={lastname,firstname,ssn}, Y={firstname} then X→ Y THEN lastname, firstname, ssn → firstname
Transitive rule: if X → Y and Y → Z then X → Z
X={ssn}, Y={deptno}, Z={deptname} and ssn→deptno and deptno→deptname
THEN ssn → deptname
Union or additive rule: if X → Y and X → Z then X → YZ
X={ssn}, Y={deptno}, Z={lastname},ssn→deptno & ssn→lastname
THEN ssn → deptno, lastname
Second Normal Form:
A relation R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R OR if every nonprime attribute A in R is not partially dependent on any key in R.
To convert from 1NF to 2NF, start with the 1NF format and write the key components on separate line and write the original key on the last line. Then each of these components will become the key in a new table. Then write the dependent attributes after each of the new keys.
Third Normal Form:
A relation schema R is in 3NF if it is in 2NF and it contains no transitive dependencies (if you have a nonprime attribute functionally dependent on another nonprime attribute)
To convert from 2NF to 3NF break off the piece(s) that are identified as transitive dependencies and store them in a separate table.
Query Optimization:
→ **The** order in which we do our SQL operations can GREATLY effect the speed with which the query is generated!
Estimates of Selectivity:
Estimates of Selectivity: the smaller the estimate is, the more desirable to use that select first (we will use this in the query optimize tree later on!)
S=ratio of tuples to satisfy a condition to the total number of record in a relation (always gives a number between 0 and 1)
Examples: *Select ... WHERE key = 34,* assume there are 100 tuples, then you would get S=1/100 = 0.01
Pipelining vs Materialization:
Pipelining: as the resulting tuples of operation are produced, they are forwarded directly to the next operation – LIMITED BY BUFFER SPACE but can improve performance.
Materialization: the results of an operation are stored in a temporary relation - could be time consuming because you have to do a write to disk! (also, sometimes unnecessary as you are going to use this file immediately anyways)
WEEK TEN:
What is a Transaction:
A **transaction** is a sequence of database operations, where the execution of the operations preserves the consistency of the database (a Logical Unit of Work)
ACID TEST: a-Atomic(everything must work, C-Consistency, I-Isolation, D-Durability.
QUESTION: Why would you want to have your log file on a different disk than your data file? **ANSWER**: Need the log to recreate the data, if the disk is gone and both the log and data are on the same disk, then you lose both!
QUESTION: Why would you want to have your log file repeated on more than 1 disk? **ANSWER**: Need to have a backup of your log file.
Concurrency Control:
Objective is to ensure serializability of transactions in multi-user database environment (interleaving of transactions where the resulting state is the same as one the states that would occur if the transactions were done serially in some order).
Schedule: A schedule S of n transactions T1, T2, ... TN is an ordering of the operations of the transactions. Each operation of an individual transaction Ti in S must appear in the same order that it appears in Ti. The scheduler interleaves the execution of database operations to ensure serializablity
Conflict: 2 operations in a schedule *conflict* if
1) they belong to different transactions, 2) they access the same data item X 3) at least one of the transactions issues a WRITE(X).
3 Main Problems: Lost Updates Uncommitted Data Inconsistent Retrievals
Scheduler: A scheduler sets the order that concurrent transactions are executed. It interweaves the operations to ensure serialization. **A schedule can use a number of methods, we will look at 3: locking, time stamping and optimistic.**
Serializability Theory: determines which schedules are "correct" and those that are not and tries to allow only correct schedules **Serial**: A schedule is serial if transactions are executed consecutively with out interleaving. Every serial schedule is considered correct
Non-serial: A schedule is non-serial if it allows interleaving of transactions
QUESTION: Why not make all schedules Serial? Slow
QUESTION: Is *result equivalence* (a non-serial schedule that produces the same final database state as a serial schedule) enough? ANSWER: No! It might have just been by accident! Example: If X starts at 100, everything is fine with both schedules but not with other numbers!
Locking: guarantees the current transaction EXCLUSIVE use of the data. **Lock Granularity: Database Level:** SLOW Table Level: Also slow, Field Level: flexible, not good tho
Lock Types: Binary Locks: an object is either Locked (1) or Unlocked(0). **Shared/Exclusive Locks**: *no lock* (unlocked) *shared lock(read) and exclusive lock*(update)
2-Phase Locking:
Ensures *serializability* but does not prevent *Deadlock*
Phase 1: The transaction acquires all required locks without unlocking any data (growing phase), once all locks have been established the transaction is at the locked point **Phase 2**: Transaction can release locks but not get new locks (shrinking phase) **Rules**:
1. No 2 transactions can have conflicting locks.
2. No unlock can precede a lock in the same transaction,
Deadlock: 2 or more transactions are waiting eternally for each other to release a lock.
Controlling Deadlock: Deadlock prevention: Transaction aborts if there is a possibility of deadlock occurring. If the transaction aborts, it must be rollback and all locks it has are released
Deadlock detection: DBMS occasionally checks for deadlock, if there is deadlock, it randomly picks one of the transactions to kill (I.e. rollback) and the other continues **Deadlock avoidance:** a transaction must obtain all it's locks before it can begin so deadlock will never occur.
Which deadlock: to useIf deadlock isn't likely use deadlock detection If deadlock is likely to happen use deadlock prevention If system response time is not a high priority use deadlock avoidance
Time Stamping:
Each transaction gets a UNIQUE GLOBAL TIMESTAMP.
The timestamp inflicts an order on the transactions.
This method DOES NOT use locks, therefore deadlocks cannot occur, however cyclic restart (abort, restart) may occur **Timestamps** are always UNIQUE and MONOTONICLY increasing, each transaction has a time stamp start time → TS(T)
Examples: T1 starts at 11:00, gets a TS(T1) → 1100
T2 starts at 11:12, gets a TS(T2) → 1112
T3 starts at 11:14, gets a TS(T3) → 1114
T1 reads X, so READTS(x) → 1100
Then T2 reads X, so READTS(X) changes to 1112
T3 wants to write to X, so WRITETS(X) → 1114
Backups: **Full Backup** (needed for catastrophic failures, old backup is used and then the log is used to bring it back to its current state) **Differential Backup** (only last changes are recorded, you can recreate database using original full backup and differential backups)
Backup of transaction log only – every single insert, update, delete are recorded.
Transaction Recovery: Write ahead log protocol: Log file is always written to BEFORE the database is actually

changed so that we can always recover (I.e. move back to a consistent state) what was done using the log file. Checkpoint in Logs: periodically records which transactions have committed and which haven't at a given time →WRITTEN TO THE LOG.