Lecture 1: Introduction	**Class Grades Break Down:**
	
	● Quizzes/Class participation – 10%
	● 3-4 programming assignments -45%
	  ● Every 3-4 weeks
	  ● weights may be uneven
	  ● Learn PHP and HTML by yourself
	  ● 25% per week deduction/maximum two weeks
	● Three exams (45%)
	  ● 15% each
	  ● in-class, closed-book exams
	  ● No final exam
	  ● No make-ups
	
	Database Management System (DBMS):
	
	- A DBMS is a powerful tool for **creating** and **managing** large amounts of data *efficiently* and allowing it to persist over long periods of time *safely*.
	
	What are the problems with file systems? (Reason behind why we converted to DBMS)
	
	- Do not support efficient access to data items
	- Do not have a query language for data items
	- Do not support concurrent control
	
	Relational Database Systems - RDBMS:
	
	Relations = Data organized as **tables**
	
	Queries could be expressed in high level language:
	
	- No need for handling complex storage structure directly
	- Greatly increased the efficiency of database programmers

	Transaction Processing A transaction is a group of database operations with certain properties. - E.g., bank transfer (withdraw + deposit) The ACID properties of transactions - A (Atomicity): all-or-nothing execution of transaction - C (Consistency): consistency constraints can not be violated - I (Isolation): each transaction must appear to be executed as if no other transaction is executing at the same time - D (Durability): the effect on the database of a transaction must not be lost once the transaction has completed The transaction processor performs the following task to ensure ACID properties: - Logging - Concurrency Control - Deadlock Resolution IMPORTANT QUERY COMMANDS: **SELECT * FROM [name of table];** - Displays all items in the table
Lecture 2&3: The Relational Model of Data	**What is a Data Model?** A data model is a notation for describing data or info. The descriptions include: 1. *Structure* of the data a. Relational Model = Tables b. Semi-structured model = tree/graphs 2. *Operations* on Data a. A limited set of queries & modifications allowed

b. The limitation allows us to describe database operations at very high levels → efficient implementations in DBMS
c. *Constraints:* limit on what data could be

Basics of Relational model:

- *Attributes*: Columns of a relation
- *Schema*: The name of a relation and the set of attributes for a relation
  - Example: Movies(title, year, length, genre)
  - The attributes are a SET, not a list
- A database consists of 1 or more relations.
- Database schema = a set of all relation schemas in the database
- Tuples: the rows of a relation (excluding the header)
  - Ex. (Star Wars, 1977, 120, SciFi)
  - It is just one row of data, a tuple has one component for each attribute of the relation
- Domains
  - Each component of each tuple must be atomic:
    - Must be a datatype (ex. Integer, string)
    - CAN NOT be a structure ex. Array, list
  - Each attribute of a relation is a domain(a particular elementary data type)

Movies (title:string, year:integer, length:integer, genre:string)

(Ask if lecture 2 slide 8 is correct)

Keys of Relations *

- A set of attributes that form a key for a relation if we do not allow 2 tuples in a relation instance to have the same values in all the attributes of the key.
- Tuples can not have the same value in all key attributes

Two ways of declaring keys by adding one of the following keywords after an attribute when declaring/altering a table

- PRIMARY KEY (the explanation of PRIMARY is deferred)
- UNIQUE

PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared as UNIQUE may have NULL's, and there may be several tuples with NULL.

SQL - Structured Query Language

- Principle language used to describe and manipulate relational databases.

Relations in SQL: three types
- Tables
- Views
- Temporary tables

Data Types

- Character strings
  - CHAR(n): a fixed-length string of up to n characters
  - VARCHAR(n): a string up to n characters
  - The difference is implementation dependent
  - Generally VARCHAR(n) is more space efficient
  - INT/INTEGER:
    - typical integer values.
    - Variants: SHORTINT (smallint for mysql/postgresql)
  - Float pointing numbers
    - **FLOAT/REAL (DOUBLE)**
    - **DECIMAL(n,d):** values consist of **n** decimal digits with the decimal point assumed to be the **d** position from the right. ➔**NUMERIC (n,d)**
  - Dates and Time
    - DATE '1948-05-14'
    - TIME '15:00:02.5'
  - Bit strings/boolean

Simple table declarations:

CREATE TABLE ...

Example:

```
CREATE TABLE Movies
(
        title           CHAR(100),
        year            INT,
        genre           CHAR(10),
        studioName      CHAR(30),
        producerC#      INT
);
```

Modifying Relation Schemas:

- How to drop a table in sql:
  - DROP TABLE [name]
- Alter table
  - ALTER TABLE [name] ADD ..
  - ALTER TABLE [name] DROP ..

Examples:

```
DROP TABLE Movies;

ALTER TABLE MovieStar ADD phone CHAR(16);

ALTER TABLE MovieStar DROP birthdate;
```

Assigning Default Values

- When tuples are created or modified, sometimes we do not have values for all components ➔NULL are assigned to the missing values
- We can change NULL to something more appropriate by using the **DEFAULT** keyword when declaring or modifying an attribute

EXAMPLES:

1. gender CHAR(1) DEFAULT '?'
2. birthday DATE DEFAULT DATE '0000-00-00'
3. ALTER TABLE MovieStar ADD phone CHAR(16)

DEFAULT ‘unlisted’

An Algebraic Query Language – Relational Algebra

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
- The result is an algebra that can be used as a *query language* for relations.

Core Relational Algebra:

- Selection: picking certain rows
- Projection: picking certain columns
- Products & Join: compositions of relations
- Renaming of relations and attributes.
- Union, intersection and difference

Selection:

R1 := σ_C(R2)

- *C* is a condition (as in “if” statements) that refers to attributes of R2.
- R1 is all those tuples of R2 that satisfy *C*.

Look @Slide3.9

Projection:

R1 := π_L(R2)

- *L* is a list of attributes from the schema of R2.
- R1 is constructed by looking at each tuple of R2, extracting the attributes on list *L*, in the order specified, and creating from those components of a tuple for R1.
- Eliminate duplicate tuples, if any.

Look @Slide3.11

Extended Projection:

- Using the same π_L operator, we allow the list *L* to

contain arbitrary expressions involving attributes:

1. Arithmetic on attributes, e.g., *A+B->C*.
2. Duplicate occurrences of the same attribute.

Look @Slide3.13

Product:

R3 := R1 X R2

- Pair each tuple t1 of R1 with each tuple t2 of R2.
- Concatenation of t1 and t2 is a tuple of R3.
- Schema of R3 is the attributes of R1 and then R2, in the order.
- But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

Natural Join:

Denoted R3 := R1 ⋈ R2.

- A *natural* join connects two relations by:
  - Equating attributes of the same name
  - Projecting out one copy of each pair of equated attributes.

Theta Join:

- R3 := R1 $\bowtie_C$ R2
  - Take the product R1 X R2.
  - Then apply σ_C to the result.
- As for σ, *C* can be any boolean-valued condition.

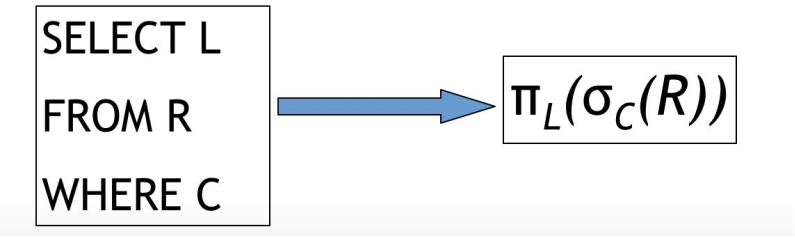
Building Complex Expressions

Three notations, just as in arithmetic:

1. Expressions with several operators.

$$R' = \pi_L(\sigma_C(R))$$

2. Sequences of assignment statements.
3. Expression trees.

Lecture 4&5: The Database Language SQL	**Select-From-Where Statements:** SELECT desired attributes FROM one or more tables WHERE condition about tuples of the table  *** In SELECT clauses:** * in the SELECT clause stands for "all attributes of this relation." Example: Using Beers(name, manf): SELECT * FROM Beers WHERE manf = 'Anheuser-Busch'; **Renaming Attributes** If you want the result to have different attribute names, use "*AS <new name>*" to rename an attribute. Example: Using Beers(name, manf): SELECT **name AS beer**, manf FROM Beers WHERE manf = 'Anheuser-Busch' **Expressions in SELECT Clauses:** Any expression that makes sense can appear as an element of a SELECT clause. Example: Using Sells(bar, beer, price): SELECT bar, beer, price*114 AS priceInYen FROM Sells;

@Slide3.17

Complex Conditions in WHERE Clause:

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.

Using Sells(bar, beer, price), find the price Joe's Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

Using Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

NULL Values:

- *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
- *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

Multi-relation Queries:

- Dealing with more than one relation aka table
- You can distinguish attributes of the same name by "<relation aka table name>.<attribute>"

Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar:

```
SELECT beer
        FROM Likes, Frequents
        WHERE bar = 'Joe''s' AND
                Frequents.drinker  =  Likes.drinker;
```

How to do multi-relation Queries:

1. Start with the **product of all the relations** in the FROM clause.
2. Apply the **selection condition** from the WHERE clause.
3. Project onto the **list of attributes and expressions** in the SELECT clause.

Slide@5.7

SELF-JOIN:

From Beers(name, manf), find all pairs of beers by the same manufacturer.

- Do not produce pairs like (Bud, Bud).
- Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).

```
SELECT b1.name, b2.name
        FROM Beers b1, Beers b2
        WHERE b1.manf = b2.manf AND
                b1.name < b2.name;
```

Subqueries:

- A parenthesized SELECT-FROM-WHERE statement is a (*subquery*)

	Subquery of the previous question: Find the beers liked by someone who frequents Joe's Bar. SELECT beer FROM LIKES, (SELECT drinker FROM Frequents WHERE bar = "Joe's)JD WHERE Likes.drinker = JD.drinker; Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe's bar charges for Bud. SELECT bar FROM Sells WHERE beer = 'Miller' AND price = (SELECT price FROM Sells WHERE bar = 'Joe''s Bar' AND beer = 'Bud'); **The IN & NOT IN Operator:** <tuple> IN (subquery) • is true if and only if the tuple is a member of the relation produced by the subquery. • Opposite: <tuple> NOT IN (<subquery>).

	Examples: Comparison
	
	`SELECT a`
	`FROM R, S`
	`WHERE R.b = S.b;`
	
	`SELECT a`
	`FROM R`
	`WHERE b IN (SELECT b FROM S);`
	
	@Slide5.21-26

Lecture 6:	**Relational Algebra on Bags:**
	- A bag or multiset is like a set, but an elemant may appear more than once.
	- Example:
	  - {1,2,3,1} is a bag
	  - {1,2,3} is also a bag that happens to be a set
	
	Why Bags?
	- The bag lang is the most important query lang for relational databases.
	
	Operations on Bags:
	- Selection
	- Projection (allows duplicated in bag operand)
	- Products and Joins
	
	Go over Slides to get info on UNION, INTERSECTION, DISTINCT, ALL…
	
	δ = eliminate duplicates from bags.
	
	τ = sort tuples.
	
	γ = grouping and aggregation.
	
	Outerjoin : avoids “dangling tuples” = tuples that do not join with anything

@Slide7.7

Aggregation Operators:

- The most important examples: SUM, AVG, COUNT(aka len), MIN, and MAX.

@Slide7.12

@Slide7.14

- Product:

  R CROSS JOIN S;

- Natural join:

  R NATURAL JOIN S;

- Example:

  Likes NATURAL JOIN Sells;

- Relations can be (parenthesized) subqueries, as well.
- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- HAVING <condition> may follow a GROUP BY clause.
  - If so, the condition applies to each group, and groups not satisfying the condition are eliminated.
- A *modification* command does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications:

1. *Insert* a tuple or tuples.
2. *Delete* a tuple or tuples.
3. *Update* the value(s) of an existing tuple or tuples.
4. To insert a single tuple:

   INSERT INTO <relation>
   VALUES (<list of values>);

	• To delete tuples satisfying a condition from some relation: DELETE FROM <relation> WHERE <condition>; Updates **To change certain attributes in certain tuples of a relation:** **UPDATE <relation>** **SET <list of attribute assignments>** **WHERE <condition on tuples>;**