**Notes 8-23-18**

Database Management System (DBMS)

* Use – Relational Algebra, SQL Queries
* Design – Functional Dependencies, Keys, Normal Forms, Access Control, Security, Data Definition
* Implementation – Query compiling, and optimization, disk layout, data structures (B-tree), failure (intrusion recovery)

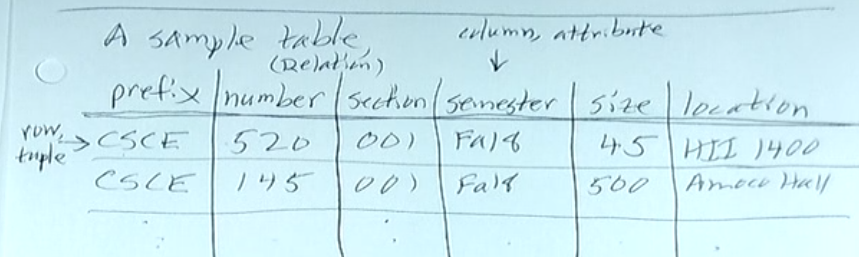
1970 – T. Codd Relational Model

* Columns called attributes
* Rows called records
* Columns are typically fixed
* Rows can be added and removed

SQL – Structured Query Language

DBMS Needs

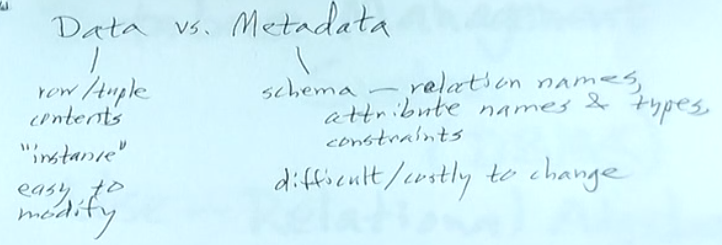
* Large datasets
* Persistent
* Safe
* Multiple users on multiple threads



A **table** is a set of rows, each row containing data for each column.

A **relation** is a set of **tuples**, each tuple containing data for each attribute.

Database Schema: A collection of relational schemas (tables) usually interrelated in some way (typically-sharing attributes)



**Notes 8-28-18**

Relational Schema – includes relation name and attribute names

|  |  |
| --- | --- |
| Database Schema | Class (prefix, courseNo, section, semester, size, location)  Course (prefix, courseNo, title)  Student (sid, name, dob, address, major)  Takes (sid, prefix, courseNo, section, semester, grade) |

SQL – 2 types of SQL commands:

* Data definition – defining the DB schema (metadata), CREATE, ALTER, DROP
* Data manipulation – SELECT (query, passive), INSERT (add a row or rows to a table), DELETE (remove row or rows from a table), DELETE (change value(s) within a row)

Relational Algebra

Means of making new relations from existing ones.

SQL 🡪 relational algebra

^-- internal

Operators & atomic operands build up expressions with these:

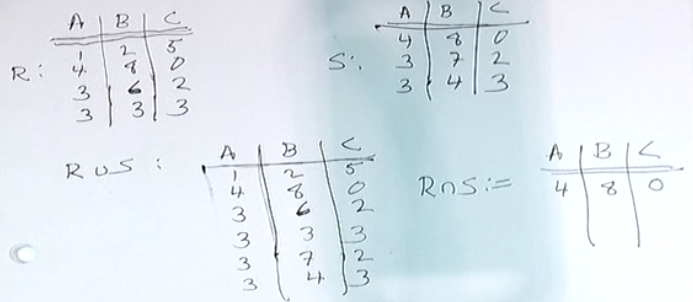
Atoms:

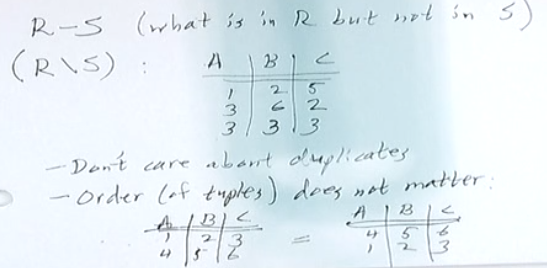
* Variables that stand for relations
* Constants (finite relations)

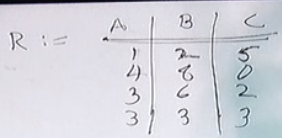
Operators:

* Set / Boolean
* Selection & Projection (restrict a relation)
* Combining Relations: cartesian product ( x ), various joins
* Renaming: schema change

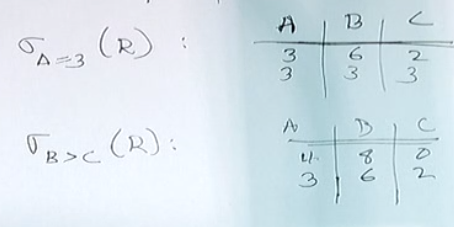
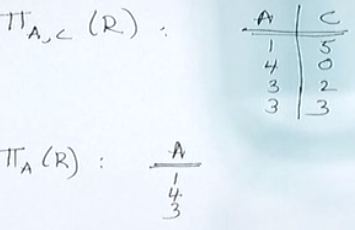
Think of a relation (for now) as a set of tuples of same type.



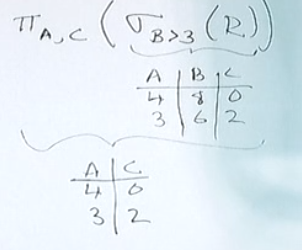




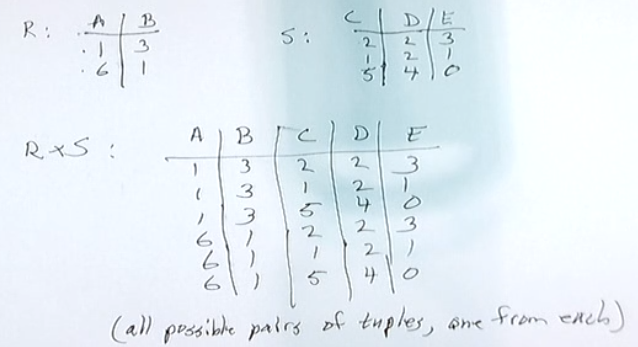
Selection Projection

Selection and Projection



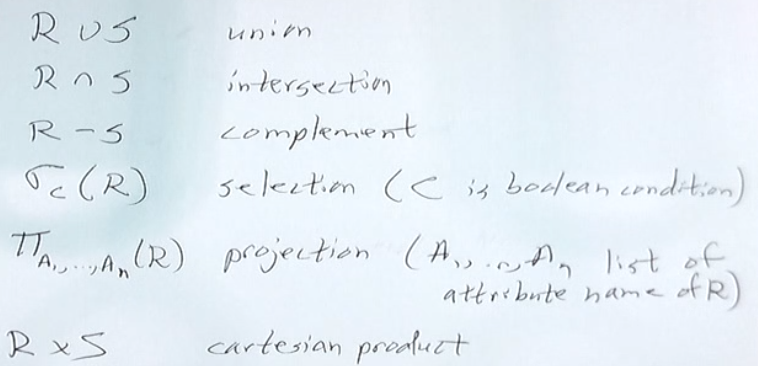
Cartesian Product



**8-30-18 Notes**

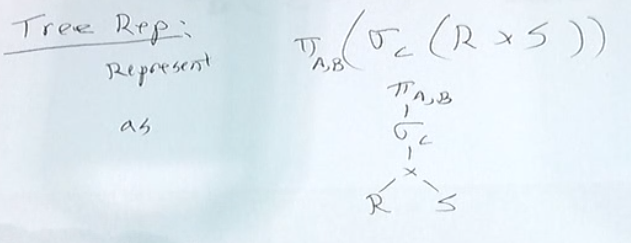
Relation Algebra Operators

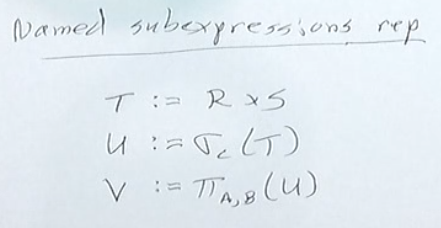
R, S relations

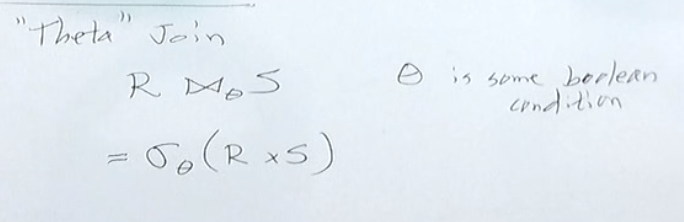


**Relational Algebra**

**Different Ways to write**





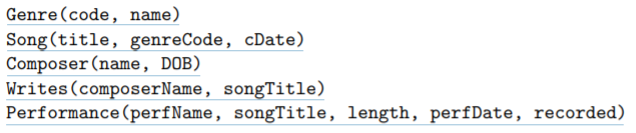


**CSCE 520, Midterm Guide I**

**Thursday October 5, 2017**

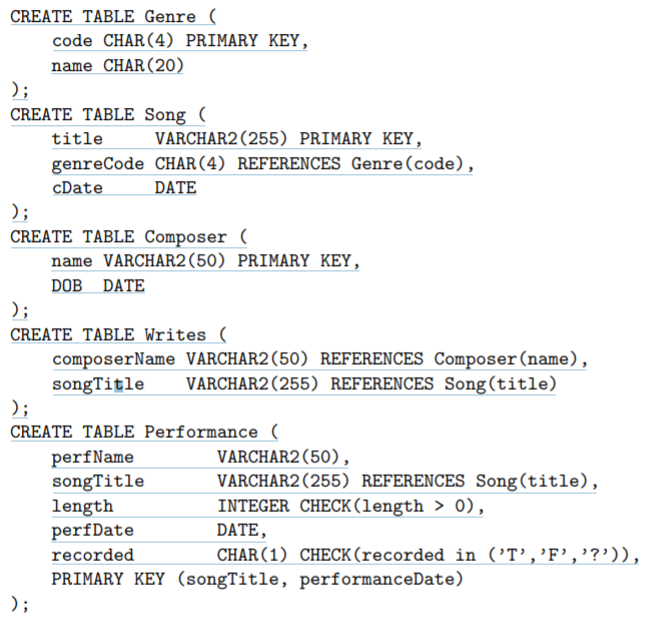
Some problems refer to the following relational database schema, related to music (and not

necessarily realistic):



Here are the SQL commands used to create the tables, above (note: ‘T’ stands for TRUE and

‘F’ stands for FALSE):



Note:

* Each song has a title that uniquely identifies it. It also has a genreCode (a code of up to three letters describing the genre, e.g., jazz, clas, rock, altr, ambt, hiho, cnty, ska, rb, etc., whose corresponding name is found in the Genre table) and a date of composition (cDate).
* Each composer has a name, which uniquely identifies her or him, and a DOB (date of birth).
* A composer may write several songs, and a song may be (co)written by any number of composers.
* Each performance is of a performer (a single person or an ensemble) performing a song and has a length in seconds and a date of performance. It may or may not be recorded. The key constraint requires that the same song is not performed twice on the same date.

1. Give expressions in relational algebra for the following queries:
2. the titles of all songs written by Cole Porter
3. the composer names and song titles of songs performed (at least once) by their composers
4. the performance dates and names of the performers of the song, “Anything Goes,” performed in 2015 and recorded
5. the names of all genres of songs performed by Keith Jarrett
6. the names of composers none of whose songs have been performed
7. the names of all entities (people or ensembles) involved with the song, “Anything Goes,” either as composers or performers
8. Write an SQL query (SELECT statement) that for each performed song returns the song title, average length of its performances, and its earliest performance date.

SELECT songTitle, avg(length), min(perfDate)

FROM Performance

GROUP BY songTitle;

1. Write a simple, concise SQL query that returns the titles of those songs that have been performed at least 100 times.

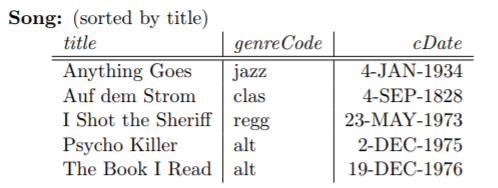
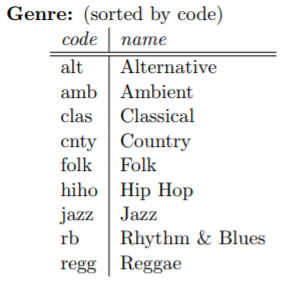
SELECT songTitle

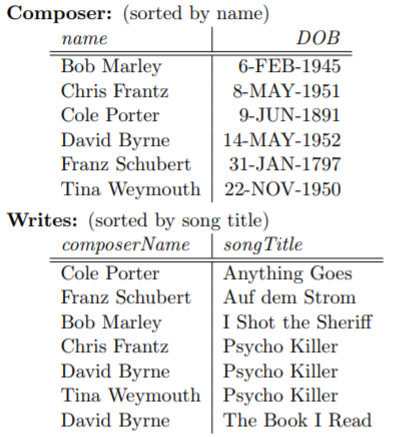
FROM Performance

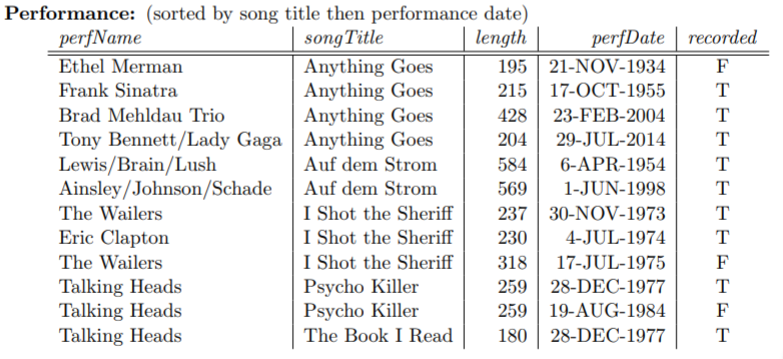
GROUP BY songTitle

HAVING COUNT(perfName) ≥ 100;

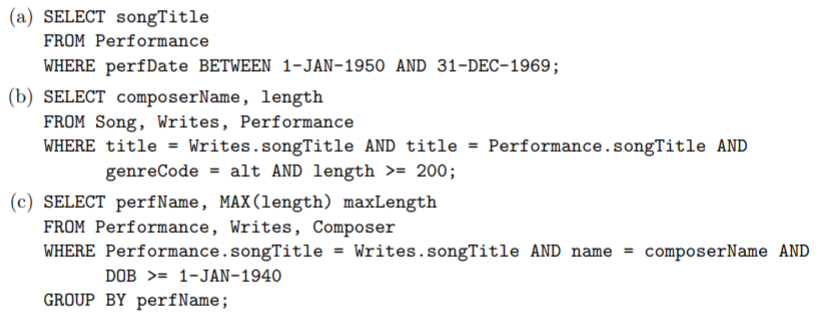
1. Suppose the tables described above are made up of the following tuples, where for ease of reference, rows are sorted by the primary key (if there is one):







What is returned by the following SQL queries? Give your answer in tabular form, including column headings. (You are free to suppress duplicate entries if you want.)



|  |  |
| --- | --- |
| composerName | length |
| Chris Frantz | 259 |
| David Byrne | 259 |
| Tina Weymouth | 259 |



|  |
| --- |
| songTitle |
| Anything Goes |
| Auf dem Storm |

|  |  |
| --- | --- |
| perfName | maxLength |
| The Wailers | 318 |
| Talking Heads | 259 |

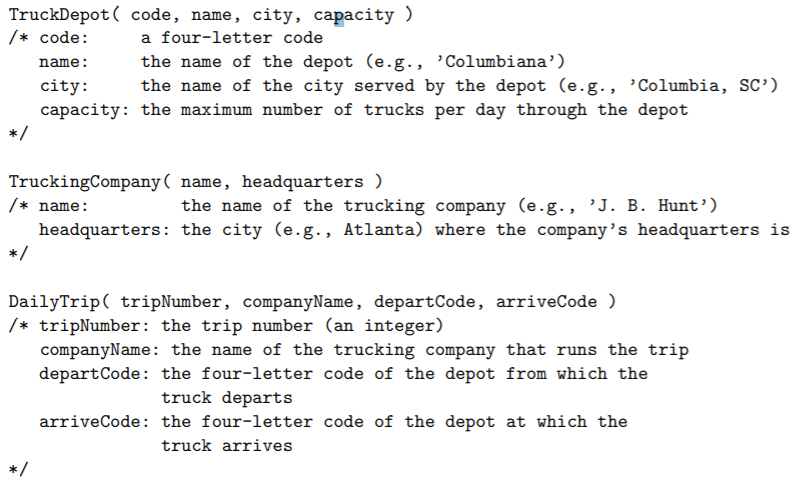


1. Let R(A, B, C, D) satisfy the FDs A → D, AB → C, C → D.
2. List all the keys for R.
3. List all BCNF violations. Give each violation in the form, S + = T, where S and T are sets of attributes, T contains an attribute not in S, but T does not contain all attributes.
4. Use the information in the last item to decompose R fully into BCNF. Your decomposition should be efficient. Give your decomposition in tree form, as depicted in class. (There are two correct answers; you should only give one of them.)

**CSCE 520, Midterm Guide 2**

**Thursday February 16, 2017**

1. In the relational model, data is represented to the user simply as collections of tuples whose entries have simple types, and only a small handful of basic operations are allowed on the data. Explain, in a sentence or two, why this is actually an advantage (over a system that allows the user direct, fine-grained control over more sophisticated data structures). The rest of the problems assume the following relations:



Assume the following constraints on the data:

**Primary Keys:** A truck depot is uniquely identified by its code. A trucking company is uniquely identified by its name. A daily trip is uniquely identified by its trip number.

**Value Constraints:** All truck depots must have positive capacity. Trip numbers for daily trips must be positive. The name of a truck depot cannot be null.

**Referential Integrity:** A company name appearing in the DailyTrip table must also appear as a name in the TruckingCompany table. Both the depart and arrive codes for a daily trip must also appear as codes in the TruckDepot table.

1. \*\*\*Give CREATE TABLE commands in SQL for the three relations above, giving types for each attribute that are reasonably appropriate and consistent, as well as expressing the given constraints.

CREATE TABLE TruckDepot (

code CHAR(4),

name CHAR(15) NOT NULL,

city CHAR(20),

capacity INT,

PRIMARY KEY (code),

CHECK (capacity >= 0)

);

CREATE TABLE TruckingCompany (

name CHAR(15),

headquarters CHAR(20),

PRIMARY KEY (name)

);

CREATE TABLE DailyTrip (

tripNumber INT,

companyName CHAR(20),

departCode CHAR(4),

arriveCode CHAR(4),

PRIMARY KEY (tripNumber),

CHECK (tripNumber >= 0),

// INDEX (companyName),

FOREIGN KEY (companyName) REFERENCES TruckingCompany (name),

// INDEX (departCode),

FOREIGN KEY (departCode) REFERENCES TruckDepot (code),

// INDEX (arriveCode),

FOREIGN KEY (arriveCode) REFERENCES TruckDepot (code)

);

1. Give expressions in relational algebra for the following queries:
2. The names of all depots that serve Columbia, SC.
3. The names of all companies running daily trips departing from Columbia, SC.

COLA = four-letter Columbia, SC code

1. \*\*\*The headquarters of all companies running daily trips arriving at Columbia, SC.

OR

1. \*\*\*The codes of all depots at which no daily trips depart nor arrive.

OR

1. Write SQL queries (SELECT statements) for each of the first three (3) items in the last problem. You have no restrictions on the form of a query, as long as what you use is described in the textbook, in class, or legal Oracle SQL. Don’t worry about duplicates; assume all relations are sets and all operations are set operations.
2. The names of all depots that serve Columbia, SC.

SELECT name

FROM TruckDepot

WHERE city = ‘Columbia, SC’;

1. The names of all companies running daily trips departing from Columbia, SC.

SELECT companyName

FROM DailyTrip

WHERE departCode = ‘COLA’;

1. The headquarters of all companies running daily trips arriving at Columbia, SC.

SELECT headquarters

FROM TruckingCompany

WHERE name IN (

SELECT companyName

FROM DailyTrip

WHERE arriveCode = ‘COLA’

);

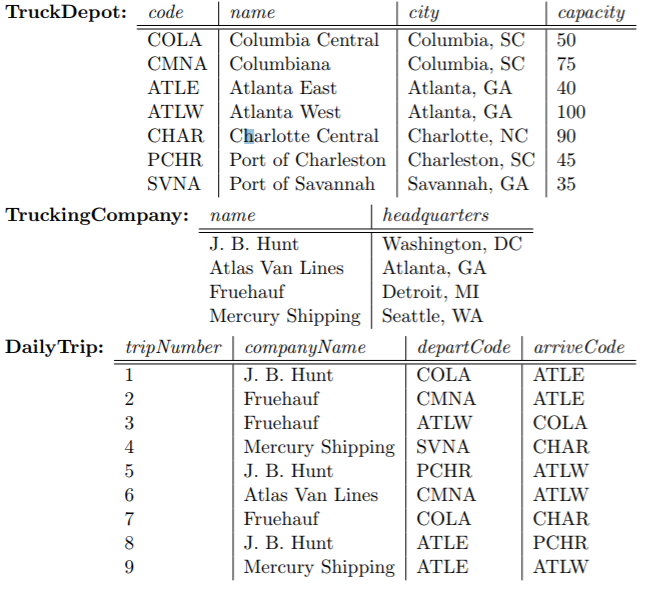
OR

SELECT headquarters

FROM TruckingCompany, DailyTrip

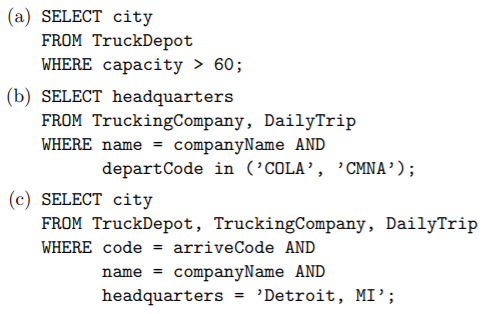
WHERE name = companyName AND arriveCode = ‘COLA’;

1. Express the constraint (in the form R = ∅, where R is some expression in relational algebra) that if some truck arrives at a depot, then some truck run by the same company (but not necessarily the same truck), must depart from the same depot.
2. Suppose the tables described above are made up of the following tuples:



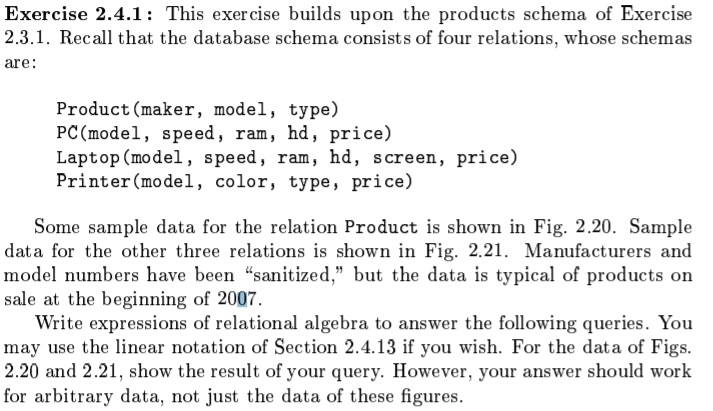
What is returned by the following SQL queries? (You are free to suppress duplicate entries if you want.)

|  |
| --- |
| city |
| Columbia, SC |
| Atlanta, GA |
| Charlotte, NC |

1. 

|  |
| --- |
| headquarters |
| Washington, DC |
| Detroit, MI |
| Atlanta, GA |

|  |
| --- |
| city |
| Columbia, SC |
| Atlanta, GA |
| Charlotte, NC |



1. What PC models have a speed of at least 3.00?
2. Which manufactures make laptops with a hard disk of at least 100GB?
3. Find the model number and price of all products (of any type) made by manufacturer B.
4. Find the model numbers of all color laser printers.
5. Find those manufacturers that sell Laptops, but not PC’s.
6. Find those hard-disk sizes that occur in two or more PC’s.