CS 455
Database Management Systems

Est. 1888
UNIVERSITY of PUGET SOUND

Department of Mathematics
and Computer Science

Lecture 3
Structured Query Language
(SQL)

Motivation

- Relational algebra is great, but...
  - DBMS are for use by common users, not just computer scientists
  - It's all "Greek" to lay users

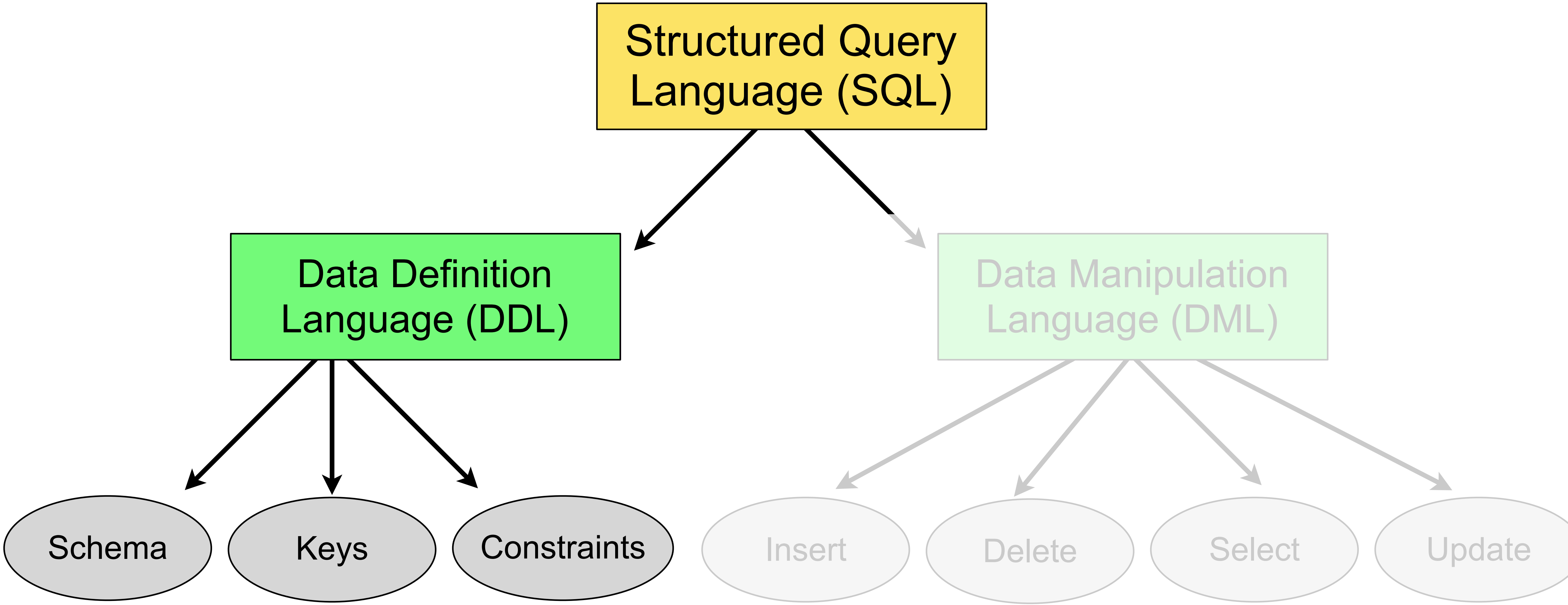
    $\sigma, \Pi, \rho, \cup, \setminus, \times, \bowtie, \ldots$

  - Need user-friendly language, that has the same *expressivity* as relational algebra

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
    - Create table
    - Drop table
    - Alter table
  - Data Manipulation Language (DML)

SQL at a Glance



(In this class, we'll use SQLite3 syntax)
... unfortunately, SQL flavors differ across implementations

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
    - Create table
    - Drop table
    - Alter table
  - Data Manipulation Language (DML)

Declaring a Relational Table

- Defining a Relation: $R(a_1, a_2, \ldots, a_n)$
- SQL Syntax:

```
create table R (

   a_1 type_1  [c_1 c_2 ...][,  -- attribute 1
   a_2 type_2  [c_1 c_2 ...],   -- attribute 2
   .
   .
   a_n type_n  [c_1 c_2 ...],]  -- attribute n

  [TC_1   -- table constraint 1
   TC_2   -- table constraint 2
   .
   .
   TC_k]   -- table constraint k
);
```

```
[...] means optional

-- is a line-comment
```

Attribute Data Types (in SQLite)

```
a_1 type_1 [c_1 c_2 ...]
```

Attribute name is followed by its data type

- `INTEGER` (Not INT!)
  - Value is a signed integer
  - 1 to 8 bytes (automatic: depending on magnitude of the number that's stored)
- `REAL`
  - Value is a double-precision floating point number
  - 8 bytes
- `TEXT`
  - Value is a text string
  - Stored as UTF8 (1-byte per char), UTF16 (2-bytes per char)
- `BLOB` (**B**inary **L**arge **Ob**ject)
  - It could be an image, PDF, video, MP3, etc.

Attribute Constraints

- Common Attribute (Column) Constraints: You can stack these constraints!
  - NOT NULL
  - UNIQUE
  - CHECK(expression)
  - PRIMARY KEY
    - If attribute type is an integer, will auto-increment if given `NULL` value.
    - *(What if your primary key is a set of two or more attributes? See "Table Constraints")*

Attribute Definition:

```
a_1 type_1 [c_1 c_2 ...]
```

Attribute constraints are optional, and stackable!

```
create table player (
    pid    INTEGER PRIMARY KEY,
    name   TEXT    UNIQUE,
    salary INTEGER NOT NULL CHECK(salary < 100000)
);
```

Example of SQLite Enforcing Constraints

```
create table player (
    pid    INTEGER PRIMARY KEY,
    name   TEXT    UNIQUE,
    salary INTEGER NOT NULL CHECK (salary < 100000)
);
```

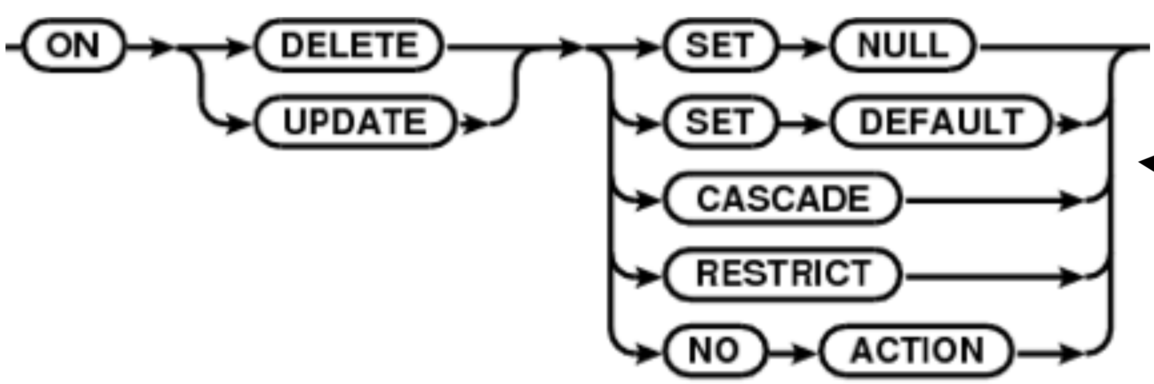
```
sqlite> INSERT INTO player VALUES (NULL, 'David', 25000);

sqlite> INSERT INTO player VALUES (1, 'Andy', 10000);
Error: PRIMARY KEY must be unique

sqlite> INSERT INTO player VALUES (NULL, 'David', 55000);
Error: column name is not unique

sqlite> INSERT INTO player VALUES (NULL, 'Tom', 55000);

sqlite> INSERT INTO player VALUES (NULL, 'Fred', 55000);

sqlite> INSERT INTO player VALUES (NULL, 'Jim', 150000);
Error: constraint failed

sqlite> select * from player;
1|David|25000
2|Tom|55000
3|Fred|55000
```

Table Constraints

- Common Table Constraints
  - PRIMARY KEY *(a_1, ..., a_n)*
  - UNIQUE *(a_1, ..., a_n)*
  - CHECK *(expression)*
  - FOREIGN KEY *(a_1, ..., a_n)* REFERENCES *R'(b_1, ..., b_n)* ...

```
TC_1    -- table constraint 1
TC_2    -- table constraint 2
.
.
TC_k     -- table constraint k
```

Again, all optional and stackable!



CASCADE is super useful!!

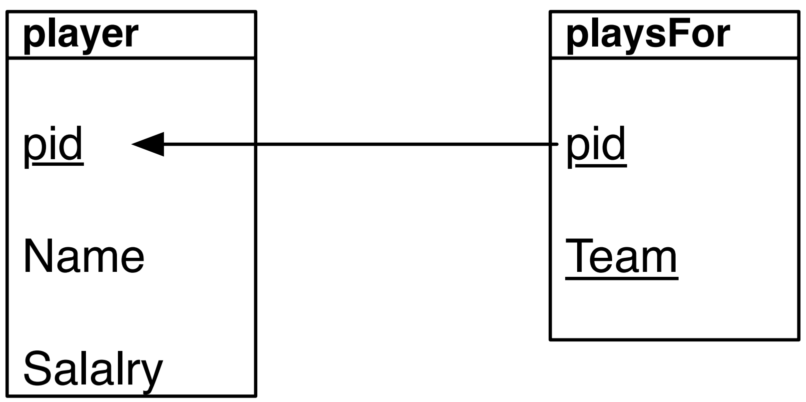
In SQLite, you have to enable foreign keys first!

```
sqlite> PRAGMA foreign_keys = ON;
```

Requires at least SQLite 3.6.19

Example Declaring Relational Tables

- Assume we have the following schema



Attribute constraints

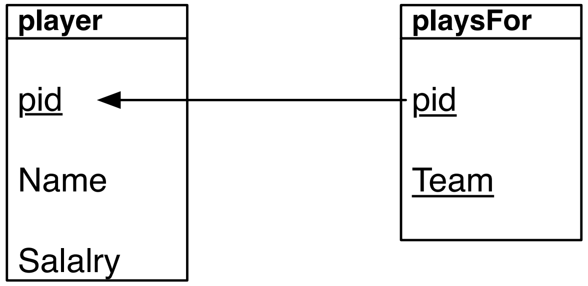
```
create table player (
pid    INTEGER  PRIMARY KEY,
name   TEXT     NOT NULL,
salary INTEGER  CHECK (salary < 100000)
);
```

Table constraints

```
create table playsFor (
pid    INTEGER,
team   TEXT,
PRIMARY KEY (pid,team),
FOREIGN KEY (pid) REFERENCES player(pid)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Foreign Key (Cascading In Action)

- Example of a Cascading Update




```
sqlite> PRAGMA foreign_keys = on;
...  -- code to create and insert some data into player and playsFor tables
sqlite> select * from playsFor;
1|Blazers
2|Blazers

sqlite> select * from player;
1|David|25000
2|Tom|55000
3|Fred|55000

sqlite> update player set pid=5 where pid=1;
```

Importance of Foreign Key Constraints

- What if the foreign key constraint *wasn't issued* when we created the table?

```
sqlite> select * from player;
1|David|55000
2|Tom|65000
3|Fred|75000

sqlite> select * from playsFor;
1|Blazers
2|Blazers

sqlite> update player set pid=6 where pid=2;

sqlite> select * from player;
1|David|55000
3|Fred|75000
6|Tom|65000

sqlite> select * from playsFor;
1|Blazers
2|Blazers -- stale!! Tom changed numbers! (Many problems now...)
```

Other Useful DDL Commands

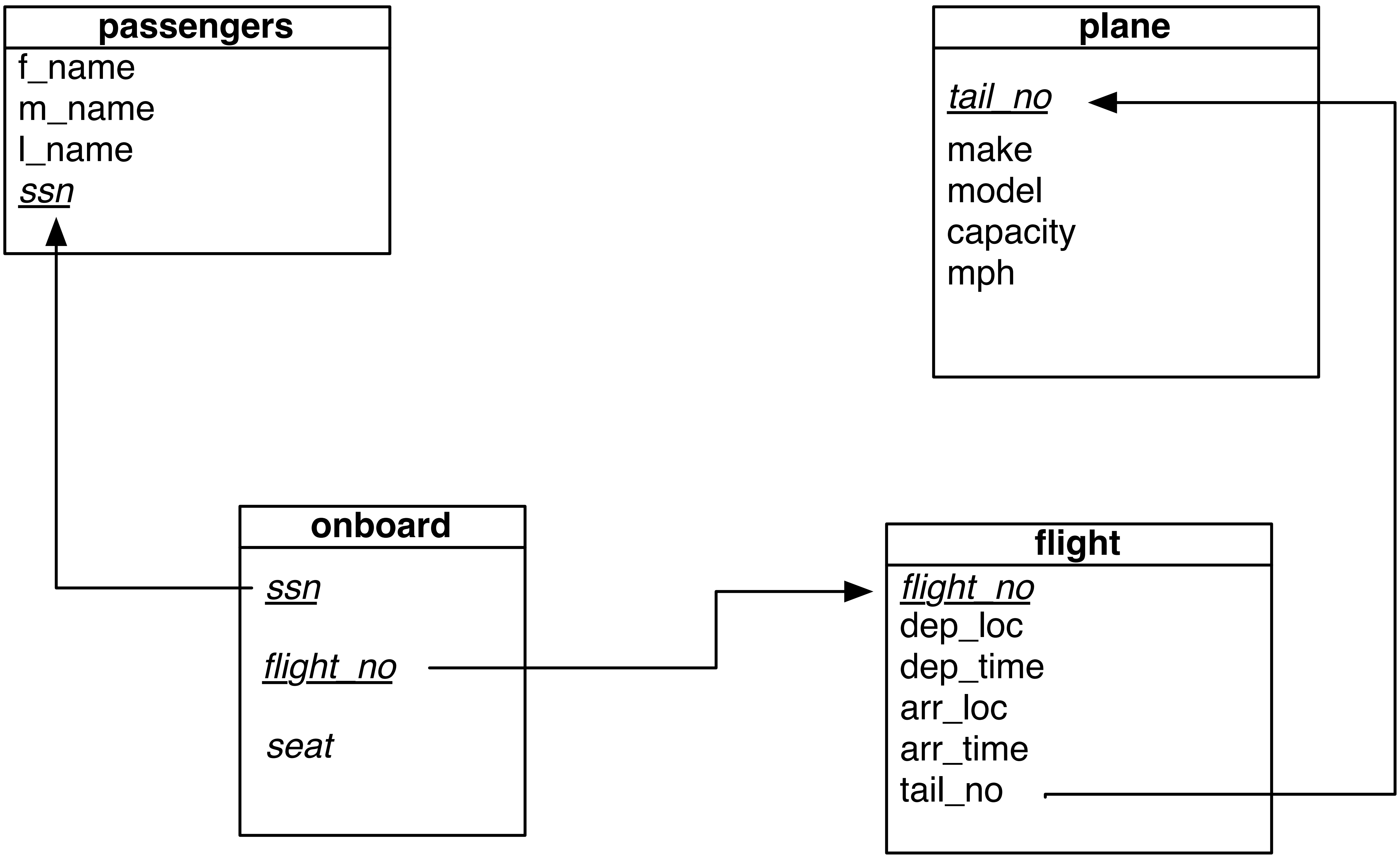
- Removing a relational table:

```
drop table [if exists] R;
```

- Changing the an existing table's properties: Look into *alter table* commands
  - For example, adding an attribute after-the-fact

```
alter table R
  ADD attr datatype c1 c2, ...
  FIRST|AFTER attr_name
```

Create This Schema in SQLite!



(A solution on course page: airport-schema.sql after today's class)

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Insert
    - Delete
    - Update
    - Select
      - from, where
      - order by
      - set operations
      - joins (implicit, explicit)
      - sub-queries
      - aggregation and grouping

SQL: Inserting Tuples

- Relational Algebra Syntax: $R \leftarrow R \cup E$
  - Example:

$$passengers \leftarrow passengers \cup \{(\text{`David'}, \text{NULL}, \text{`Chiu'}, \text{`888-88-8888'})\}$$

- SQL Syntax:

```
INSERT INTO R VALUES (v1, v2, ..., vn);
```

- Example:

```
INSERT INTO passengers VALUES
       ('David', NULL, 'Chiu', '888-88-8888');
```

SQL: Deleting Tuples

- Relational Algebra Syntax: $R \leftarrow R \setminus E$
- Example:

$$player \leftarrow player - \sigma_{Salary>60000}(player)$$

- SQL Syntax:

```
DELETE FROM R [WHERE C];

-- everything in [...] is optional
-- That means WHERE clause is optional!
--     C is assumed true if not given.
```

- Example:

```
DELETE FROM player WHERE Salary > 60000;

DELETE FROM player; -- OMG What did I just do??
```

SQL: Updating Tuples

- Relational Algebra Syntax: $R \leftarrow \Pi_{E_1,...,E_k}(R)$

- SQL Syntax:

```
UPDATE R SET a_1=e_1[, ..., a_k=e_k] [WHERE C];

-- e_i are SQL expressions
-- C is assumed true if not given
```

- Example:

```
UPDATE passengers SET m_name='John' WHERE ssn='888-88-8888';

UPDATE players SET salary=(1.04*salary); -- I meant to leave off
where clause..
```

Select-From-Where (SFW)

- Recall this common R.A. expression: $\Pi_{a_1,...,a_k}(\sigma_c(R))$
- In SQL, this projection-selection takes on a very common form:
  - Often referred to as an *SFW-query*

```
SELECT [distinct] a_1[,a_2,...,a_k]|*
FROM R1[,R2,...,Rn]
[WHERE C];

-- Once again, C is assumed true if not given

-- Note: Instead of listing all attributes, you can use '*'
to mean all attributes

-- Note 2: multiple relations mean cross product!

-- Note 3: distinct keyword?
```

Comparison Operators

- Common comparison operators for the `WHERE` clause:

```
expr1 = expr2 (or expr1 == expr2 in SQLite)
expr1 < expr2
expr1 > expr2
expr1 <= expr2
expr1 >= expr2
expr1 <> expr2 (or expr1 != expr2 in SQLite)

attr IS [NOT] NULL
attr [NOT] BETWEEN expr1 AND expr2

attr [NOT] LIKE expr
attr [NOT] GLOB regexp (not supported by most other DBMS)

attr [NOT] IN expr (later)
attr [NOT] EXISTS (later)
```

Let's Run Some SFW Queries

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Timothy		Lovejoy	555-55-5555
Joe	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

plane

tail_no	make	model	capacity	mph
0	Boeing	747	525	570
1	Boeing	747	525	570
2	Airbus	A350	270	580
3	McDonnel Douglas	DC10	380	610

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
720	Springfield, IL	7:15	Chicago, IL	7:45	3
86	Columbus, OH	16:00	Portland, OR	22:00	3
303	New York, NY	12:30	Miami, FL	13:00	1
1142	Paris, France	15:00	Munich, Germany	17:40	0
5932	Hartford, CT	9:00	Phoenix, AZ	12:00	0
495	Miami, FL	10:45	Austin, TX	13:30	1
5031	Akron, OH	14:20	Hartford, CT	16:45	2

```
sqlite> .read airport-schema.sql   -- creates the tables and constraints
sqlite> .read airport-populate.sql -- populates the tables with above data
```

(Find these files on the course page!)

Joins in SQL

Nice-Looking Bowties	Relational Algebra	SQL Equivalent
	$R_1 \bowtie R_2$	R1 NATURAL JOIN R2
	$R_1 \bowtie_\theta R_2$	*(next slide)*
	$R_1 ⟕ R_2$	R1 LEFT OUTER NATURAL JOIN R2
	$R_1 ⟖ R_2$	R1 RIGHT OUTER NATURAL JOIN R2
	$R_1 ⟗ R_2$	R1 FULL OUTER NATURAL JOIN R2

(Didn't I say these were "extended" relational operators?)

Inner-Join (Also known as Theta Join)

- The \theta-join is defined: $\sigma_\theta(R_1 \times R_2) = R_1 \bowtie_\theta R_2$

- If you declare more than one relation in the FROM clause, it performs a *cross product* on all declared relations

$R_1 \times R_2$	<= same as =>	`FROM R_1, R_2`

- The rest? Just use WHERE conditions to formulate the Join

```
SELECT * FROM R1, R2 WHERE theta;
```

- Or, the inner-join syntax:

```
SELECT * FROM R1 JOIN R2 ON theta;
```

Rename Operator

- Syntax
  - First form: $\rho_{R_2}(R_1)$
  - Meaning: Renames relation R_1 to R_2

  - Second form: $\rho_{(b_1,...,b_n)}(R_1)$
  - Meaning: Renames $R_1(a_1, \ldots, a_n)$ to $R_1(b_1, \ldots, b_n)$

  - Third form: $\rho_{R_2(b_1,...,b_n)}(R_1)$
  - Meaning: Renames $R_1(a_1, \ldots, a_n)$ to $R_2(b_1, \ldots, b_n)$

Renaming Relational Tables (1st Form)

- Relational Algebra Syntax
  - **First form**: $\rho_{R_2}(R_1)$
  - Meaning: Renames relation R_1 to R_2
- SQL syntax:

```
SELECT ...
FROM   R1 as X [, R2 as Y, R3 as Z, ...]
[WHERE  C];

-- Renames R_1 to A' [R_2 to B', ... ]
-- R_1 can no longer be referred to in the query
```

```
sqlite> select * from passengers as A where A.l_name='Flanders';
f_name      m_name      l_name      ssn
----------  ----------  ----------  -----------
Ned         T           Flanders    777-77-7777

sqlite> select * from passengers as A where passengers.l_name='Flanders';
Error: no such column: passengers.l_name
```

Renaming Attributes (2nd Form)

- Relational Algebra Syntax
  - **Second form**: $\rho_{(b_1, \ldots, b_n)}(R_1)$
  - Meaning: Renames R_1's attributes from $R_1(a_1, \ldots, a_n)$ to $R_1(b_1, \ldots, b_n)$
- SQL syntax:

```
SELECT a1 as b1 [a2 as b2, ...]
FROM   ...
[WHERE  C];

-- Renames a1 to b1, a2 to b2, etc.
```

```
sqlite> select f_name as F, l_name as L from passengers where l_name='Simpson';

F           L
----------  ----------
Homer       Simpson
Bart        Simpson
Lisa        Simpson
```

Renaming Relations and Attributes (3rd Form)

- Relational Algebra Syntax
  - **Third form**: $\rho_{R_2(b_1,...,b_n)}(R_1)$
  - Meaning: Renames $R_1(a_1, \ldots, a_n)$ to $R_2(b_1, \ldots, b_n)$
- SQL Syntax:

```
SELECT a1 as b1 [a2 as b2, ...]
FROM   R1 as X [, R2 as Y, R3 as Z, ...]
[WHERE  C];

-- But SQL has different take...
```

```
sqlite> select mph as speed from plane as pl where pl.mph > 500;

speed
----------
570
570
580
610
```

Operationally, SQLite splits the 3rd-form into two parts...
(1) Apply 1st-form immediately, then
(2) Apply 2nd-form

$$\rho_{(speed)}(\Pi_{mph}(\sigma_{pl.mph>500}(\rho_{pl}(planes))))$$

Tricky Rename Example from Before

- Find all pairs of passengers that share last names:
  - (Homer, Bart), (Lisa, Bart), (Lisa, Homer)
- Let's try...

```
select p1.f_name, p2.f_name
from passengers as p1, passengers as p2
where p1.l_name = p2.l_name;
```

```
f_name      f_name
----------  ----------
Homer       Bart
Homer       Homer
Homer       Lisa
Bart        Bart
Bart        Homer
Bart        Lisa
Lisa        Bart
Lisa        Homer
Lisa        Lisa
Frank       Frank
Robert      Robert
Ned         Ned
Frank       Frank
```

(Problem 1: Don't want the same first names)

Tricky Rename Example from Before (Cont.)

- Find all pairs of passengers that share last names:
  - (Homer, Bart), (Lisa, Bart), (Lisa, Homer)
- Let's try...

```
select p1.f_name, p2.f_name
from   passengers as p1, passengers as p2
where  p1.l_name = p2.l_name
       and
       p1.f_name != p2.f_name;
```

```
f_name       f_name
----------   ----------
Homer        Bart
Homer        Lisa
Bart         Homer
Bart         Lisa
Lisa         Bart
Lisa         Homer
```

(Problem 2: Want the same combinations just once!)

Tricky Rename Example from Before (Cont.)

- Find all pairs of passengers that share last names:
  - (Homer, Bart), (Lisa, Bart), (Lisa, Homer)
- Let's try...

```
select p1.f_name, p2.f_name
from   passengers as p1, passengers as p2
where  p1.l_name = p2.l_name
       and
       p1.f_name > p2.f_name;
```

```
f_name      f_name
----------  ----------
Homer       Bart
Lisa        Bart
Lisa        Homer
```

(When comparison operators like > and < are used with text, it means...?)

Imposing Order on Results

- Databases do not guarantee any ordering on the returned results
  - But we can impose an ordering using the following *optional* clause:

Any *SELECT ... FROM ... WHERE ...* query

```
SFW

[ORDER BY a_1 [DESC] [, a_2 [DESC], ..., a_k [DESC]]];

-- Sorts results by the given attribute(s) before returning

-- Sorts by a_1 first, then a_2, then a_3, ...

-- DESC keyword sorts results in descending order
```

Imposing Order on Results (Cont.)

```
sqlite> select * from plane order by make, tail_no;

tail_no     make        model       capacity    mph
----------  ----------  ----------  ----------  ----------
3           Airbus      A350        270         580
5           Airbus      A380        200         500
1           Boeing      747         525         570
2           Boeing      747         525         570
4           McDonnel D  DC10        380         610
```

```
sqlite> select * from passengers order by l_name, f_name desc;

f_name      m_name      l_name      ssn
----------  ----------  ----------  -----------
Ned         T           Flanders    777-77-7777
Frank       NULL        Lovejoy     555-55-5555
Robert      N           Quimby      666-66-6666
Frank       NULL        Ryerson     333-33-3333
Lisa        G           Simpson     222-22-2222
Homer       J           Simpson     111-11-1111
Bart        H           Simpson     444-44-4444
```

Set Operators

- We've already seen cross-product through implicit joins, but what about these?
  - Remember, to apply any of these operations, the two relations must be *compatible*

- SQL Syntax:
  - `SFW1 UNION [ALL] SFW2`
  - `SFW1 INTERSECT [ALL] SFW2`
  - `SFW1 EXCEPT [ALL] SFW2`

  (ALL keyword retains duplicates!)

- Examples:
  1. Get Lisa and Ned's passenger info
  2. Get all last names and plane model numbers
  3. Get everyone but Bart

For Reference

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Timothy		Lovejoy	555-55-5555
Joe	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

plane

tail_no	make	model	capacity	mph
0	Boeing	747	525	570
1	Boeing	747	525	570
2	Airbus	A350	270	580
3	McDonnel Douglas	DC10	380	610

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
720	Springfield, IL	7:15	Chicago, IL	7:45	3
86	Columbus, OH	16:00	Portland, OR	22:00	3
303	New York, NY	12:30	Miami, FL	13:00	1
1142	Paris, France	15:00	Munich, Germany	17:40	0
5932	Hartford, CT	9:00	Phoenix, AZ	12:00	0
495	Miami, FL	10:45	Austin, TX	13:30	1
5031	Akron, OH	14:20	Hartford, CT	16:45	2

Example with the [ALL] Keyword

```
sqlite> select l_name from passengers union
        select model from plane;
l_name
----------
747
A350
A380
DC10
Flanders
Lovejoy
Quimby
Ryerson
Simpson
```

```
sqlite> select l_name from passengers union all
        select model from plane;
l_name
----------
Simpson
Simpson
Simpson
Lovejoy
Quimby
Flanders
Ryerson
747
747
A350
DC10
A380
```

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Insert
    - Delete
    - Update
    - Select From Where
      - order by
      - joins (implicit, explicit)
      - set operations
        - sub-queries
      - aggregation and grouping

Subqueries

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

- Recall that queries could be nested in relational algebra. We saw this example:
  - Find the last names of all passengers not onboard any flights:

$$\Pi_{l\_name}(\sigma_{ssn \notin \boxed{\Pi_{ssn}(onboard)}}(passenger))$$

(1) Nested query returns the set of ssn for all passengers onboard any flight

Nested Projection Result =

ssn
555-55-5555
111-11-1111
777-77-7777
666-66-6666
444-44-4444
777-77-7777

Subqueries (Cont.)

passengers

f_name	m_name	l_name	ssn
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	flight_no	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

- Recall that queries could be nested in relational algebra. We saw this example:
  - Find the last names of all passengers not onboard any flights:

$$\Pi_{l\_name}\left(\boxed{\sigma_{ssn \notin \Pi_{ssn}(onboard)}(passenger)}\right)$$

(2) Now we select all the tuples from passengers where ssn is <u>NOT IN</u> the set returned by the nested query (1)

Select Result =

f_name	m_name	l_name	ssn
Lisa	G	Simpson	222-22-2222
Frank		Ryerson	333-33-3333

Subqueries (Cont.)

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

- Recall that queries could be nested in relational algebra. We saw this example:
  - Find the last names of all passengers not onboard any flights:

$$\boxed{\Pi_{l\_name}(\sigma_{ssn \notin \Pi_{ssn}(onboard)}(passenger))}$$

(3) Project lastname of all tuples selected in step (2).

Final Projection Result =

l_name
Simpson
Ryerson

Subqueries: In, Not-in

UNIVERSITY of PUGET SOUND

- What was the key operator? $\in$ $\notin$
  - Checking to see if tuples were *in* or *not-in* the nested query result
- We can use these with nested queries in the WHERE clause

```
SELECT ...
FROM ...
WHERE a [NOT] IN (SWF);

-- selects all tuples such that attribute a is in or not
in the results of a nested SWF query
```

```
sqlite> select l_name from passengers where ssn NOT IN (select ssn from onboard);

l_name
----------
Simpson
Ryerson
```

Subquery Example 2

passengers

f_name	m_name	l_name	ssn
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	flight_no	seat
555-55-5555	6	32F
111-11-1111	2	2B
222-22-2222	4	1F
777-77-7777	5	25A
333-33-3333	3	25A
666-66-6666	1	30C
444-44-4444	7	30C
777-77-7777	3	25A
555-55-5555	4	25A

plane

tail_no	make	model	capacity	mph
1	Boeing	747	525	570
2	Boeing	747	525	570
3	Airbus	A350	270	580
4	McDonnel Douglas	DC10	380	610
5	Airbus	A380	200	500

For SQL Subquery Lectures

Notes:
Airbus A380 not flying any flights

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
1	Springfield, IL	7:15	Chicago, IL	7:45	4
2	Columbus, OH	16:00	Portland, OR	22:00	4
3	New York, NY	12:30	Miami, FL	13:00	2
4	Paris, France	15:00	Munich, Germany	17:40	1
5	Hartford, CT	9:00	Phoenix, AZ	12:00	1
6	Miami, FL	10:45	Austin, TX	13:30	2
7	Akron, OH	14:20	Hartford, CT	16:45	3

- Find the f_name, l_name of passengers sitting in 25A on any flight.
  - Let’s use both join and sub-query

Subquery Example 3

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	6	32F
111-11-1111	2	2B
222-22-2222	4	1F
777-77-7777	5	25A
333-33-3333	3	25A
666-66-6666	1	30C
444-44-4444	7	30C
777-77-7777	3	25A
555-55-5555	4	25A

plane

tail_no	make	model	capacity	mph
1	Boeing	747	525	570
2	Boeing	747	525	570
3	Airbus	A350	270	580
4	McDonnel Douglas	DC10	380	610
5	Airbus	A380	200	500

For SQL Subquery Lectures

Notes:
Airbus A380 not flying any flights

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
1	Springfield, IL	7:15	Chicago, IL	7:45	4
2	Columbus, OH	16:00	Portland, OR	22:00	4
3	New York, NY	12:30	Miami, FL	13:00	2
4	Paris, France	15:00	Munich, Germany	17:40	1
5	Hartford, CT	9:00	Phoenix, AZ	12:00	1
6	Miami, FL	10:45	Austin, TX	13:30	2
7	Akron, OH	14:20	Hartford, CT	16:45	3

- Find the f_name of passengers sitting in 25A on any flight.
  - Can we still answer this query using only joins?

Subquery Example 4

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	6	32F
111-11-1111	2	2B
222-22-2222	4	1F
777-77-7777	5	25A
333-33-3333	3	25A
666-66-6666	1	30C
444-44-4444	7	30C
777-77-7777	3	25A
555-55-5555	4	25A

plane

tail_no	make	model	capacity	mph
1	Boeing	747	525	570
2	Boeing	747	525	570
3	Airbus	A350	270	580
4	McDonnel Douglas	DC10	380	610
5	Airbus	A380	200	500

For SQL Subquery Lectures

Notes:
Airbus A380 not flying any flights

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
1	Springfield, IL	7:15	Chicago, IL	7:45	4
2	Columbus, OH	16:00	Portland, OR	22:00	4
3	New York, NY	12:30	Miami, FL	13:00	2
4	Paris, France	15:00	Munich, Germany	17:40	1
5	Hartford, CT	9:00	Phoenix, AZ	12:00	1
6	Miami, FL	10:45	Austin, TX	13:30	2
7	Akron, OH	14:20	Hartford, CT	16:45	3

- Find the planes that are <u>not</u> flying into Germany
  - Join version, Subquery version

Subqueries: Exists, Not Exists

$\exists$ $\nexists$

- In addition to in/not-in, we have exists/not-exist to test whether the subquery returns a set with elements or an empty set.

```
outer-SFW [NOT] EXISTS (inner-SFW);

-- selects all tuples such that the nested SWF query
returns anything (or nothing)
```

This is an expensive operation! Here's what it does:

```
for each tuple t returned in the outer SFW,
  run the inner SFW
  if inner SFW [doesn't] returns anything //[NOT] EXISTS?
     retain t
  else
     discard t
```

Subquery Example 5

passengers

f_name	m_name	l_name	ssn
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	flight_no	seat
555-55-5555	6	32F
111-11-1111	2	2B
222-22-2222	4	1F
777-77-7777	5	25A
333-33-3333	3	25A
666-66-6666	1	30C
444-44-4444	7	30C
777-77-7777	3	25A
555-55-5555	4	25A

plane

tail_no	make	model	capacity	mph
1	Boeing	747	525	570
2	Boeing	747	525	570
3	Airbus	A350	270	580
4	McDonnel Douglas	DC10	380	610
5	Airbus	A380	200	500

For SQL Subquery Lectures

Notes:
Airbus A380 not flying any flights

flight

flight_no	dep_loc	dep_time	arr_loc	arr_time	tail_no
1	Springfield, IL	7:15	Chicago, IL	7:45	4
2	Columbus, OH	16:00	Portland, OR	22:00	4
3	New York, NY	12:30	Miami, FL	13:00	2
4	Paris, France	15:00	Munich, Germany	17:40	1
5	Hartford, CT	9:00	Phoenix, AZ	12:00	1
6	Miami, FL	10:45	Austin, TX	13:30	2
7	Akron, OH	14:20	Hartford, CT	16:45	3

- Find the fastest plane in the fleet

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Insert
    - Delete
    - Update
    - Select
      - from, where
      - order by
      - set operations
      - joins (implicit, explicit)
      - sub-queries
      - aggregation and grouping

Aggregation and Grouping

- Recall the R.A. Syntax $_{g_1,...,g_j}\mathcal{G}_{f_1(a_1),...,f_k(a_k)}(R)$

Empoyees

ENO	Dept	Country	Name	Wage
0	A	US	John	50
1	A	China	Lynn	75
3	B	US	Ross	60
7	C	US	Julia	95
8	B	China	David	25
9	A	China	Ned	65

- Q1: Get the count of all employees
- Q2: Get the count of all employees, min wage, and max wage
- Q3: Get the count of all employees, min wage, and max wage by department and country

(Let's do each in R.A. first)

Aggregation and Grouping in SQL

```
SELECT f_1([distinct] a_1)[, ..., f_k([distinct] a_k),
                          g_1, ..., g_j,
                          a_1, ..., a_n]
FROM R_1 [, R_2, ...]
WHERE C
[GROUP BY g_1, g_2, ..., g_j [HAVING C']];
```

Aggregation Functions

Groups

Other attributes

Groups

Group cond.

```
sqlite> select avg(capacity), avg(mph) from plane group by make;
avg(capacity)  avg(mph)
-------------  ----------
235.0          540.0
525.0          570.0
380.0          610.0
```

**** What happened to my group, *'make'*? *****
In SQL, you have to project the groups too
(You did not have to do this with R.A.)

```
sqlite> select avg(capacity), avg(mph), make from plane group by make;
avg(capacity)  avg(mph)    make
-------------  ----------  ----------
235.0          540.0       Airbus
525.0          570.0       Boeing
380.0          610.0       McDonnel D
```

HAVING Clause: Selecting Only Certain Groups

```
SELECT f_1([distinct] a_1)[, ..., f_k([distinct] a_k),
                           g_1, ..., g_j,
                           a_1, ..., a_n]
FROM R_1 [, R_2, ...]
WHERE C
[GROUP BY g_1, g_2, ..., g_j [HAVING C']];
```

Return only groups "Having" condition C'

- Find all departments, countries, and their average wages whose average wage is < $50

- Find the department that pays the highest average wage

Empoyees

ENO	Dept	Country	Name	Wage
0	A	US	John	50
1	A	China	Lynn	75
3	B	US	Ross	60
7	C	US	Julia	95
8	B	China	David	25
9	A	China	Ned	65

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
    - Create table
    - Drop table
    - Alter table
  - Data Manipulation Language (DML)
    - SFW, Rename, Joins
    - Ordering results
    - Subqueries
    - Grouping/Aggregation
  - Views
- Conclusion

Views

- Often, it is undesirable for all users to see entire database
- For example, security and privacy concerns...
  - Health records (HIPAA)
  - Student records (FERPA)
- More desirable:
  - We only want some users to see parts of the database that certain users are authorized to see.

Views (Cont.)

- Give certain users only a read-only *logical view* of the entire database!
  - Users cannot use write operations on the view!
  - insert, delete, update prohibited!

- Syntax:

```
create view V as SFW;

drop view V;
```

Example of Views in SQLite

- Hide passenger SSN from flight attendants
- Show only first/last name, flight number, and seat assignment
- Sorted by last name
- Call this view passinfo

passengers

f_name	m_name	l_name	*ssn*
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	*flight_no*	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

---

passinfo

f_name	l_name	*flight_no*	seat
Ned	Flanders	5932	25 A
Ned	Flanders	303	25 A
Frank	Lovejoy	495	32 F
Robert	Quimby	720	30 C
Bart	Simpson	5031	30 C
Homer	Simpson	86	1 D

The View

Syntax: Creating Views

passengers

f_name	m_name	l_name	ssn
Homer	J	Simpson	111-11-1111
Bart	H	Simpson	444-44-4444
Lisa	G	Simpson	222-22-2222
Frank		Lovejoy	555-55-5555
Robert	N	Quimby	666-66-6666
Ned	T	Flanders	777-77-7777
Frank		Ryerson	333-33-3333

onboard

ssn	flight_no	seat
555-55-5555	495	32 F
111-11-1111	86	1 D
777-77-7777	5932	25 A
666-66-6666	720	30 C
444-44-4444	5031	30 C
777-77-7777	303	25 A

passinfo

f_name	l_name	flight_no	seat
Ned	Flanders	5932	25 A
Ned	Flanders	303	25 A
Frank	Lovejoy	495	32 F
Robert	Quimby	720	30 C
Bart	Simpson	5031	30 C
Homer	Simpson	86	1 D

- SQL:

```
create view passinfo as
      select f_name,l_name,flight_no,seat from passengers
      natural join onboard order by l_name;
```

Views in SQLite

```
sqlite> create view passinfo as select f_name,l_name,flight_no,seat
from passengers natural join onboard order by l_name;

sqlite> select * from passinfo;
f_name      l_name      flight_no   seat
----------  ----------  ----------  ----------
Ned         Flanders    3           25A
Ned         Flanders    5           25A
Frank       Lovejoy     6           32F
Robert      Quimby      1           30C
Homer       Simpson     2           2B
Bart        Simpson     7           30C

sqlite> delete from passinfo where l_name='Simpson';
Error: cannot modify passinfo because it is a view
```

Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
    - Create table
    - Drop table
    - Alter table
  - Data Manipulation Language (DML)
    - SFW, Rename, Joins
    - Ordering results
    - Subqueries
    - Grouping/Aggregation
- Conclusion

In Conclusion...

- SQL first appeared in IBM System-R (1976)
  - Now the standard relational data query language
  - Same expressivity as relational algebra, but user-friendly
- Two parts:
  - DDL: Deals with structure of database
  - DML: Deals with data
- Next: What constitutes good DB design?
  - Entity-Relationship (ER) Model
  - Normalization