Midterm Review

ER Definitions

Entity: App "object" distinguishable from other objects

Attribute: Information that describes the entity

Attribute *domain*: Range of permissible values

e.g. integers 1-20, 20 character strings, timestamp

Entity Set: Collection of entities with same attributes

Relationship: Association between entities

Relationship Set: Collection of similar relationships

Keys

Minimal set of attributes that uniquely identify an entity

May be multiple candidate keys

- e.g. User: both uid and email may be unique

May involve multiple attributes

- e.g. Class identified by both number and section

Primary key: designated unique identifier

Most entities have a key (except weak entities)

Diagram: Underlined

Weak Entity

Entity without a key: attributes are not unique

Identifying relationship distinguishes them

- e.g. Wall Post: User who posted it
- e.g. Song: Album where first released

Partial key: attributes that identify the weak entity, for a given owning entity

- e.g. Wall Post: Timestamp attribute
- e.g. Song: Title Name attribute

Diagram: Dashed underline

More ER Definitions

Aggregation:

Allows a relationship in relationships

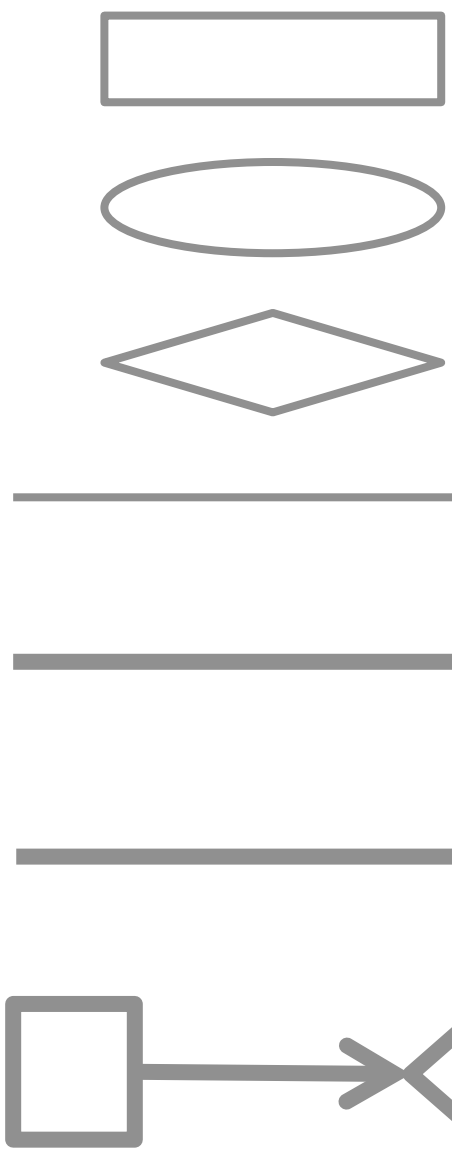
IS A / Class hierarchy:

An entity is a sub-type of another entity

Has all attributes of the parent type, plus others

Covering: permitted to be a plain parent type?

Overlap: can be more than one subtype?



Entity

Attribute

Relationship

At most one (key constraint)

At least one (participation con.)

Exactly one

Weak Entity

Class Hierarchy

Aggregation

Relational Terminology

Formal Name	Synonyms
Relation	Table
Tuple	Row, Record
Attribute	Column, Field
Domain	Type
Cardinality	# of tuple
Degree	# of attributes

Integrity Constraints (ICs)

def: a condition that is true for *any* instance of the database

Often specified when defining schema
DBMS enforces ICs at all times

An instance of a relation is legal if it satisfies all declared ICs
Programmer doesn't have to worry about data errors!
    e.g., data entry errors

Don't Repeat Yourself (DRY)
PostgreSQL documentation great resource
    www.postgresql.org/docs/8.1/static/ddl-constraints.html

Candidate Keys

Set of fields is a *candidate key (or just Key)* for a relation if:

1. Two distinct valid tuples cannot have same values
2. This is **not** true for any subset of the key (minimal)

If (2) is false, called a superkey what's a trivial superkey?

If >1 candidate keys in relation, admin assigns primary key:
Used to identify tuples elsewhere in the database

sid is key for Students
is name a key?
what is (sid, gpa)?

Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j's primary key (logical pointer)

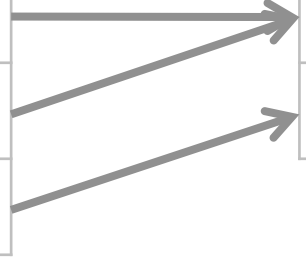
```
CREATE TABLE Enrolled(
    sid int,   cid int,   grade char(2),
    PRIMARY KEY (sid, cid),
    FOREIGN KEY (sid) REFERENCES Students
)
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis



Referential Integrity

A database instance has *referential integrity* if all foreign key constraints are enforced no dangling references

Examples where referential integrity is not enforced

- HTML links
- Yellow page listing
- Restaurant menus
- Some relational databases!

ER → Relational: 1. No constraints

- Entity: table with primary key, no foreign keys
- Relationship: table with:
  - Primary key = set of primary key of related entities
  - Foreign keys: each primary key of related entities
- Aggregation
  - Convert inner relationship (aggregated relationship)
  - Treat table for aggregated relationship as entity
- ISA: tables for each type or just subtypes

2. Add constraints

At most one: On relationship, add: UNIQUE(entity key)
OR combine tables (permit NULL)

Exactly one: Combine tables; NOT NULL constraints

At least one: Not representable

Eliminate redundancy:

PRIMARY KEY (a, b) + UNIQUE (b) = PRIMARY KEY (b)

(same primary key? Can combine tables)

Perfect translation

For all possible database instances:

- Constraints violated in ER are violated in relational
- Constraints violated in relational are violated in ER
- If ER doesn't violate, neither should relational
- If relational doesn't violate, neither should ER

Some diagrams cannot be perfectly translated

(e.g. at least one constraints)

How to check a translation?

For each ER constraint:

1. a. Find example that violates ER constraint

1. b. Verify it violates relational version

2. a. Find example that passes ER constraint

2. b. Verify it passes relational version

"~~Proof~~ by example": No guarantee; effective

Relational Algebra Overview

Core 5 operations

- PROJECT (π)
- SELECT (σ)
- UNION (U)
- SET DIFFERENCE (-)
- CROSS PRODUCT (x)

Additional operations

- RENAME (p)
- INTERSECT (∩)
- JOIN (⋈)

Project

$$\pi_{<attr1,\ldots>}(A) = R_{result}$$

Extract desired fields (subset of columns)

Schema is subset of input schema in the projection list

$\pi_{<a,b,c>}(A)$ has output schema (a,b,c) w/ types carried over

Project

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

$\pi_{age}(S2) =$

age
21
88

Where did all the rows go?
Real systems typically don't remove duplicates. Why?

Select

$$\sigma_{<p>}(A) = R_{result}$$

Select subset of rows that satisfy condition *p*

Won't have duplicates in result. Why?

Result schema same as input

Select

S1

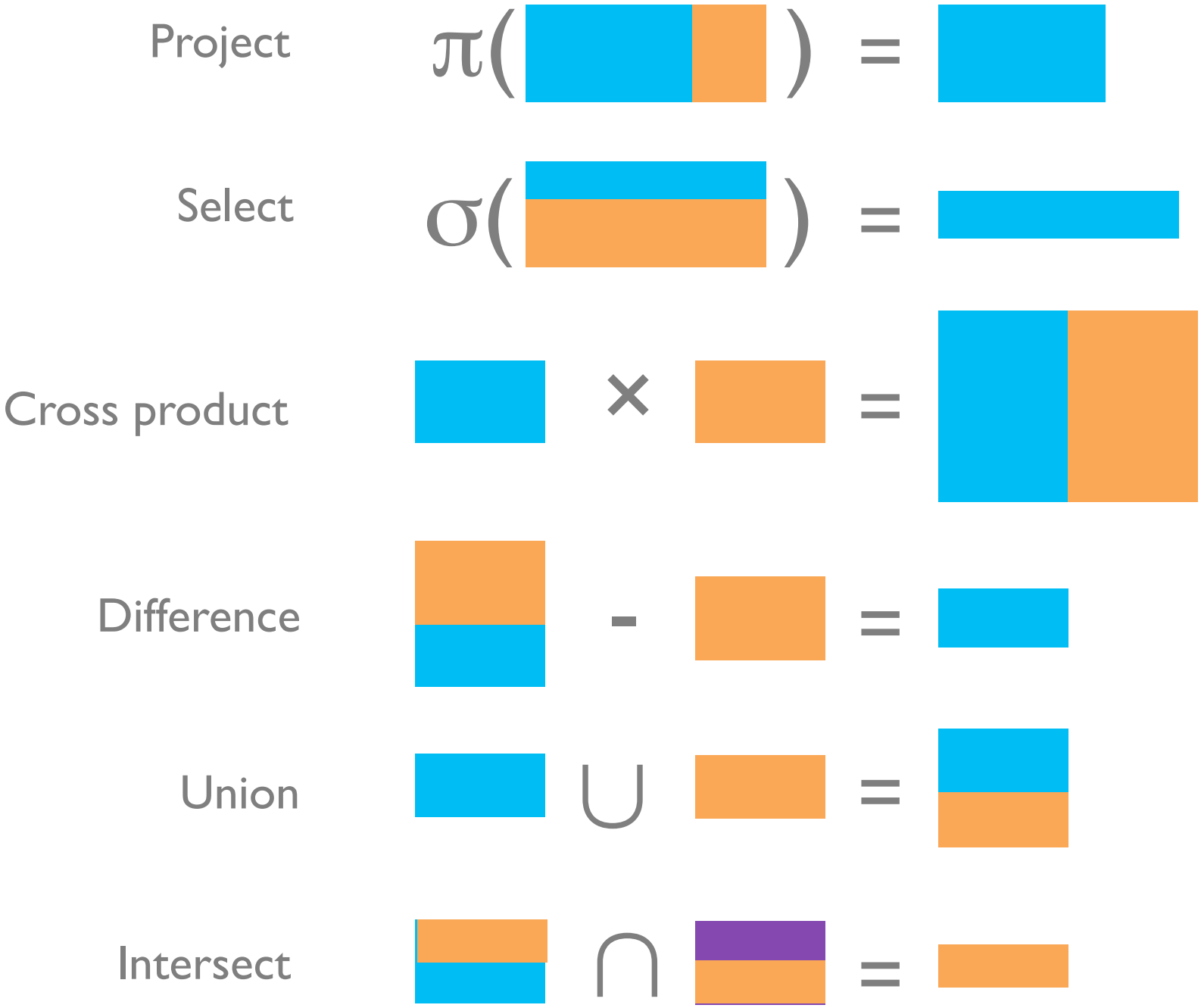
sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

$\sigma_{age<30}\ (S1) =$

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21

$\pi_{name}(\sigma_{age<30}\ (S1)) =$

name
eugene
barak


Project
π( ) =
Select
σ( ) =
Cross product
× =
Difference
- =
Union
∪ =
Intersect
∩ =

Union, Set-Difference

$$A \text{ op } B = R_{result}$$

A, B must be *union-compatible*

- Same number of fields
- Field i in each schema have same type

Result Schema borrowed from first arg (A)

- Student(sid int, age int) U Class(cid int, max int) = R_{result}(sid int, age int)

Union

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

S1 ∪ S2 =

sid	name	gpa	age
1	eugene	4	20
4	aziz	3.2	21
5	rusty	3.5	21
3	trump	2	88
2	barak	3	21

Set-Difference

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

S1 - S2 =

sid	name	gpa	age
1	eugene	4	20

Intersect

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

$S1 \cap S2 =$

sid	name	gpa	age
2	barak	3	21
3	trump	2	88

Cross-Product

$$A(a_1,\ldots,a_n) \times B(a_{n+1},\ldots,a_m) = R_{result}(a_1,\ldots,a_m)$$

Each row of A paired with each row of B

Result schema:

Combine A and B's fields, inherit names

If name conflict: need positions

Cross-Product

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

R1

sid	rid	day
1	101	10/10
2	102	11/11

S1 x R1 =

(sid)	name	gpa	age	(sid)	rid	day
1	eugene	4	20	1	101	10/10
2	barak	3	21	1	101	10/10
3	trump	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11
3	trump	2	88	2	102	11/11

Rename

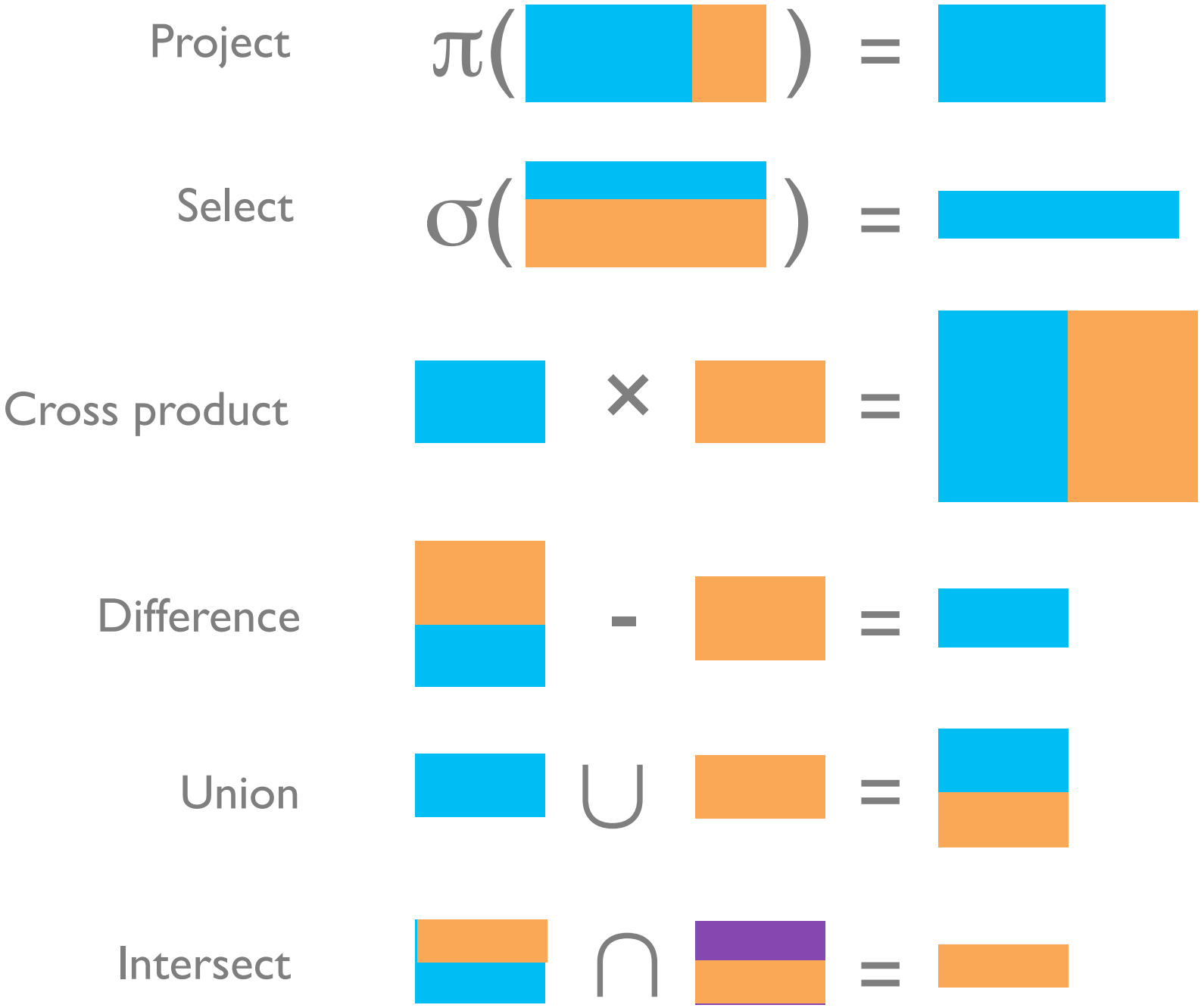
ρ(<new_name>(<mappings>), Q)

Explicitly defines/changes field names of schema

ρ(C(1 → sid1, 5 → sid2), S1 x R1)

C =

sid1	name	gpa	age	sid2	rid	day
1	eugene	4	20	1	101	10/10
2	barak	3	21	1	101	10/10
3	trump	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11
3	trump	2	88	2	102	11/11


Project
π( ) =
Select
σ( ) =
Cross product
× =
Difference
- =
Union
∪ =
Intersect
∩ =

theta (θ) Join

$$A \bowtie_c B = \sigma_c(A \times B)$$

Most general form

Result schema same as cross product

Often *far* more efficient to compute than cross product

theta (θ) Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

R1

sid	rid	day
1	101	10/10
2	102	11/11

$S1 \bowtie_{S1.sid \leq R1.sid} R1 =$

(sid)	name	gpa	age	(sid)	rid	day
1	eugene	4	20	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11

Equi-Join

Common case where the condition is attribute equality

$$A \bowtie_{attr} B = \pi_{\text{all attrs } \textit{except } B.attr}(A \bowtie_{A.attr = B.attr} B)$$

Result schema only keeps *one copy* of equality fields

Natural Join ($A \bowtie B$):

Equi-join on *all* shared fields (fields w/ same name)

Equi-Join

SI

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

RI

sid	rid	day
1	101	10/10
2	102	11/11

$SI \bowtie_{sid} RI =$

sid	name	gpa	age	rid	day
1	eugene	4	20	101	10/10
2	barak	3	21	102	11/11

SQL for tests

Don’t need to get syntax 100% correct

(that is what the database is for)

HOWEVER: If your answer is ambiguous, we will assume the worst

Expressions you should know

Math: +, -, *, /

Text: x LIKE ‘a%’

Aggregate: COUNT, MAX, MIN, AVG, SUM

SQL features you should know

Simple single table select/project:

```
SELECT a, b, c FROM table WHERE …
```

Multi-table joins:

```
SELECT * FROM a, b
WHERE a.id = b.aid AND …
```

SQL features you should know

Nested queries (careful with number of results)

UNION [ALL]/ INTERSECT/EXCEPT

GROUP BY/HAVING: Aggregation

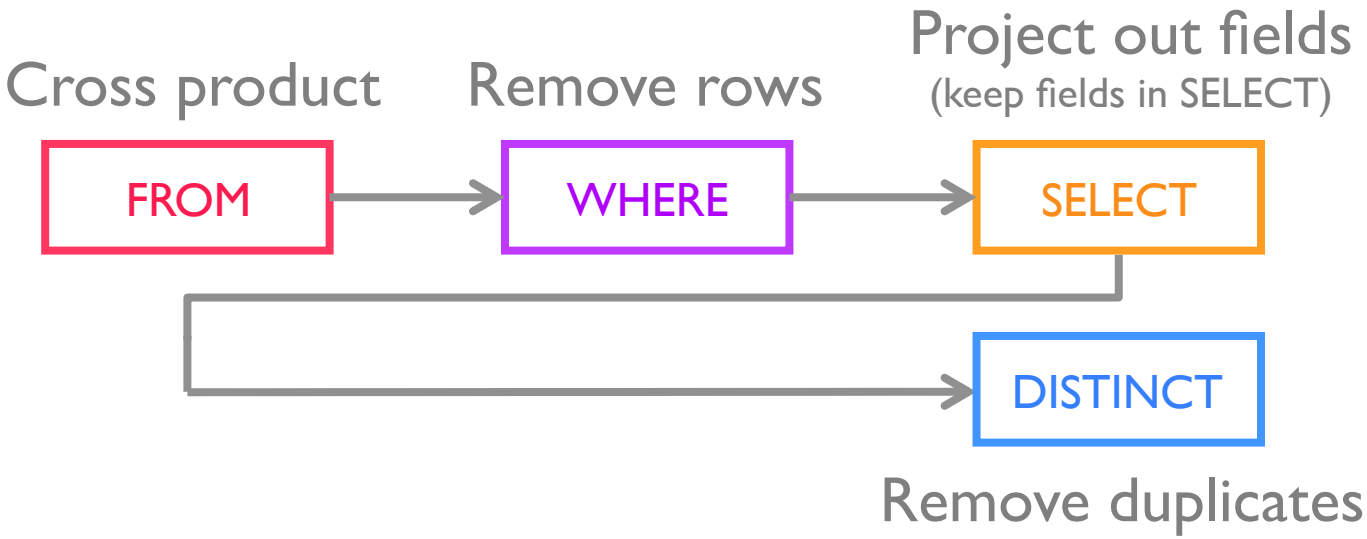
ORDER BY/LIMIT

[LEFT|RIGHT|FULL] OUTER JOIN

CHECK constraints

SQL: Simple Query Evaluation

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualification*



Not how actually executed! Above is likely very slow

Simple multi-table queries

```
SELECT S1.name, S1.age, S2.name, S2.age
FROM   Sailors AS S1, Sailors AS S2
WHERE  S1.age > S2.age
```

Multi-set VS Set semantics!

```
SELECT S.sid
FROM   Sailors AS S, Reserves AS R
WHERE  S.sid = R.sid
```

VS

```
SELECT DISTINCT S.sid
FROM   Sailors AS S, Reserves AS R
WHERE  S.sid = R.sid
```

Nested Queries

```
SELECT   S.sid
FROM     Sailors S
WHERE    EXISTS (SELECT   *
                 FROM     Reserves R
                 WHERE    R.bid = 101 AND
                          S.sid = R.sid)
```

Conceptual model:

```
   for each Sailors tuple
      run the subquery and evaluate
qualification
```

SET Comparison Operators

x IN *r*: True if value *x* appears in *r*

EXISTS *r*: True if relation *r* is not empty (NOT EXISTS)

x (operator) ANY r: True if x (operator) is true for any row in r

E.g. x IN r is equivalent to x = ANY r

x (operator) ALL r: True if x (operator) is true for all rows in r

E.g. x NOT IN r is equivalent to x <> ALL r

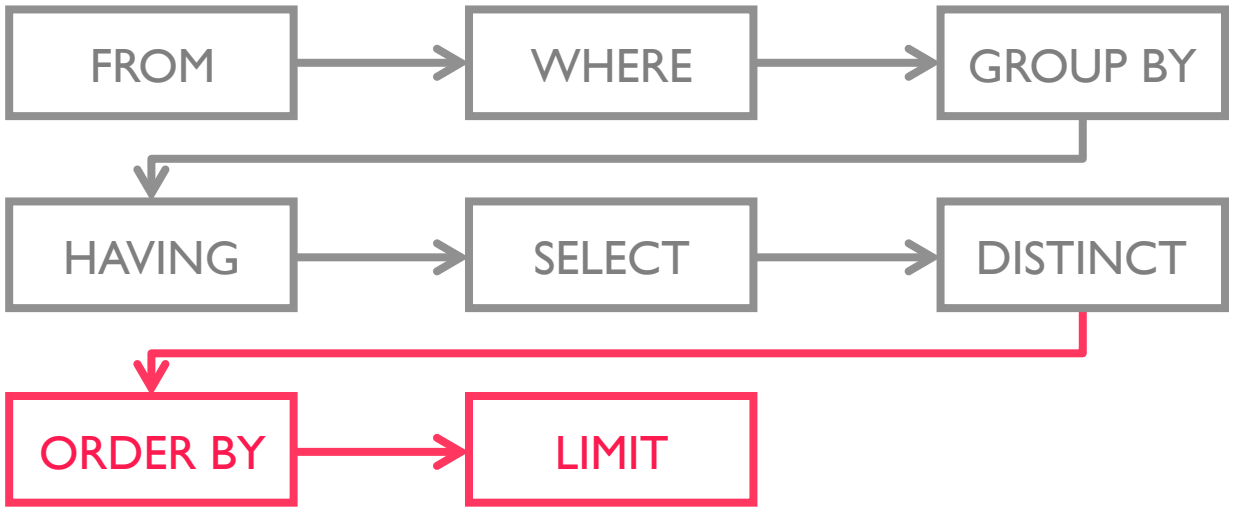
Aggregates: 1 result from set

```
SELECT AVG(S.rating), MAX(S.age)
FROM   Sailors S
```

```
        avg          | max
---------------------+-----
 5.6666666666666667 |  39
(1 row)
```

ORDER BY, LIMIT

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
ORDER BY  order-list
LIMIT     limit-expr [OFFSET offset-expr]
```



NULL

(null > 0) = null

(null + 1) = null

(null = 0) = null

(null AND true) = null

null is null = true

AND	T	F	NULL
T	T	F	NULL
F	F	F	F
NULL	NULL	F	NULL

FULL OUTER JOIN

```
SELECT  s.sid, s.name, r.bid
FROM    Sailors s Full OUTER JOIN Reserves r
ON      s.sid = r.sid
```

Sailors

sid	name	rating	age
1	Eugene	7	22
2	Luis	2	39
3	Ken	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
4	109	9/20

Result

	sid	name	bid
	1	Eugene	102
	2	Luis	102
Left	3	Ken	NULL
Right	NULL	NULL	109