Final Review Session

Relational Algebra

- projection (π) ex. $\pi_{id, citation}(stops)$
- selection (σ) ex. $\pi_{id, citation}(\sigma_{age < 30}(stops))$
- renaming (ρ) ex. $\rho_{stops\_data(id \rightarrow person\_id)}(stops)$
- cartesian / cross product ($\times$) ex. $\sigma_{stops.location = zips.location}(stops \times zips)$

SQL Basics

SELECT [DISTINCT] (5)
FROM (1)
WHERE (2)
GROUP BY (3)
HAVING (4)
ORDER BY (6)
LIMIT (7)

Joins

- theta join ($\bowtie_\theta$) : cartesian → selection condition
  - ↳ ex. $A \bowtie_{A.column2 > B.column2} B$
- equi join : cartesian → equality condition
  - ↳ ex. $A \bowtie_{A.column2 = B.column2} B$
- natural join : when two relations share same attribute; dupes removed
  - ↳ ex. $A \bowtie B$ (will not have two location columns)

Set Notation

Union ($A \cup B$)
↳ A OR B
Intersection ($A \cap B$)
↳ A AND B
Difference ($A - B$ or $A \backslash B$)
↳ A NOT B

ex.

equi join
equi join

rename student → a
rename teachers → b

$\pi_{name, instructor\_for}(\sigma_{class = instructor\_for}(student \times \sigma_{b.area = "datascience"}(teachers)))$

Subqueries

Def: Parenthesized SQL statement used within an outer SQL statement

1. As a scalar (singular value)
2. To check if a query is not empty via the EXISTS clause

Window Functions

Def: fxn that uses values from more than one row to produce a value (potentially diff) for each row
↳ agg functions similar but produce single value for all rows

uses: cumulative sum, determine rank within group, normalize a row using a group agg

3 components of window function:

1. PARTITION BY: groups the rows so that functions will be calculated within these groups instead of the entire set of rows
2. ORDER BY: sort rows in window frame if the order of rows is important (eg. running totals)
3. RANGE: specify the window frame's relation to current row

ex.)

```
SELECT student_name, order_id, sum(price)
OVER (PARTITION BY student_id) AS total_spent
FROM orders
ORDER BY order_id;
```

ORDER BY order_id

Summary

Views
- ↳ virtual: output not stored, computed on demand
  - CTEs: used within larger queries
  - materialized: stored on disk, periodically refreshed

Strings

Regex

```
REGEXP_REPLACE(source, pattern, replacement[, flags])
```

flags: optional
- "g": all matches

Substrings

```
SUBSTRING(string, start, length)
SUBSTRING(string, regex)
```

Other

```
STRPOS(string, substring)
```

↳ ex. STRPOS('snickerdoodle', 'doodle')

Sampling

```
ORDER BY RANDOM LIMIT n;
TABLESAMPLE BERNOULLI(p);
TABLESAMPLE system(p);
```

↳ by memory page NOT row

DML (Data manipulation Language)

Insertion

INSERT INTO Relation VALUES (<list>)

↳ ex. INSERT INTO stops VALUES (5000, 'Asian', 24, 'west Oakland')

ex. INSERT INTO stops (id, race, age, location) VALUES (5000, 'Asian', 24, 'west Oakland')

INSERT INTO Relation (< Subquery >)

Updating

UPDATE Relation

SET < list of new column value assignments >

WHERE < condition >

↳ex.) UPDATE stops
SET age = 18
WHERE age IS NULL ;

Deleting

DELETE FROM < relationname >

DELETE FROM <relationname > WHERE < condition>

↳ ex. age < 18

DDL (Data Definition Language)

CREATE: defines schema

DROP: deletes both schema and instance

ALTER: redefines schema

ex.)
```
CREATE TABLE Zips(
  location VARCHAR(20)
  zipcode INTEGER
);
```

Constraints

PRIMARY KEY vs. UNIQUE

- there can only be one primary key
- there can be many UNIQUE constraints
- NO attribute of PK can be null
- Attributes within UNIQUE can be null, can be tuples w/ NULL values for UNIQUE attributes

Foreign Keys

a field in one table, that refers to the PRIMARY KEY in another table

↳ ex.) CREATE TABLE Cast_info
(person_id INT FOREIGN KEY REFERENCES Actor(actor_id)

↳ex.)
```
location VARCHAR(20) NOT NULL,
arrest BOOLEAN DEFAULT False,
PRIMARY KEY (stopID)
UNIQUE (person ID, stop Time)
```

Performance Tuning

Indexing: speeds up reads on a specific key so queries are faster, reduce scans

```
CREATE INDEX nameIdIndex ON Actor(name,id);
DROP INDEX idNameIndex;
```

high cardinality = high number of distinct values

↳ low cardinality good when you know data is clustered on the indexed attribute

↳ CLUSTER Stops ON ageIndex

Query Optimization

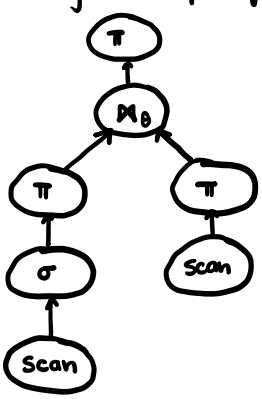
useful for reducing the size of relational instances and avoiding blow-up of joins

predicate pushdown: pre-select rows before join

projection pushdown: pre-select columns join

ex.)

```
SELECT C.name, S.name, S.prevMargin
FROM Candidate AS C
    INNER JOIN states AS S
        ON C.homestateID != S.sid
WHERE S.population > 5000000 AND
```



Performance Joins

nested loop join

↳ for loops, for every tuple of R and S check if it matches

sort-merge join

↳ walk down the two runs in their sorted order, merge matching tuples together

hash join

↳ tuples hashed into each bucket should be read from both R and S at same time

Data Preparation

gaussian outliers : removes values more than K z-scores away

windsorizing : replace outlier values with those just before those removed on a specified boundary

median absolute deviation (MAD) : $MAD(X) = \text{median}(|X_i - \tilde{X}|)$

$1\ SD = 1.4826\ MADs$

Hampel x84 : filtering outliers $K \cdot 1.4826$ MADs away

levenshtein distance : count # of insertions, deletions, mutations (replacements) btw strings

blocking + matching :

↳ blocking : q-gram blocking where we block for substrings of length q

large values of q leads to bad recall

small values of q leads to decreased precision

↳ matching : clustering entities into equivalent classes, compute distance, and clustering close values by minimizing distance

imputation : method to fill in NULLS

↳ interpolation

- run
- run start
- run end
- run size
- run rank

Multivitamin #2 Q5

int stricter than float

Region	Route	Avg_pokemon_HP	Number_caught	Legendary_caught?
Kalos	21	75	593	0
Hoenn	Route 121	62	3960	0
Alola	11	81	2363.0	1
NULL	115	60	NULL	0
Johto	32	94	4735	0
Hoenn	Route 108	121	1294	1
NULL	Route 34	55	2735.0	0
Kanto	8	78	NULL	0
Sinnoh	205	83	NULL	1
Kalos	17	70	9948	0

Q5.2
3 Points

We decide to use MDL to determine the optimal type for column 4, `Number_caught`. Assume floats are 32 bits, ints are 16 bits, booleans are 1 bit, and a character is 8 bits.

Q5.2.1
Suppose the initial type chosen is integer. What would be the result of the sum in the MDL equation?

144

Q5.2.2
Now we choose the float type. What would be the result of the sum in the MDL equation?

224

5.2.1

16 + 16 + 32 + 0 + 16 + 16 + 32 + 16

= 144

5.2.2

32 + 32 + 32 + 0 + 32 + 32 + 32 + 0 + 0 + 32

= 224

↳ choose integer since MDL wants to minimize calculated sum

Date	Week	Client	Project	Time_spent (Hrs)
Mon 3/8/21	7	Cal	Website	3
Mon 3/8/21	7	UCLA	Newsletter	2
Tues 3/9/21	7	Cal	Website	2
Tues 3/9/21	7	UCLA	Newsletter	4
Wed 3/10/21	7	Cal	Hiring	15
Wed 3/10/21	7	UCLA	Website	2
Thurs 3/11/21	7	Cal	Merch Launch	4
Thurs 3/11/21	7	UCLA	Hiring	10
Fri 3/12/21	7	Cal	Newsletter	2
Fri 3/12/21	7	UCLA	Scheduling	24
Mon 3/15/21	8	Cal	Scheduling	37
Mon 3/15/21	8	UCLA	Merch Launch	2

Q4.3.1
We execute the following pivot operation:

Pivot(Client, Project, Time_spent(Hrs), sum)

How many columns will be in the resulting pivot table? (NOTE: Do not treat row labels as a column)

Pivot(<column 1>, <column 2>, <column3>, <Agg function>)

↑ columns ↑ rows ↑ input

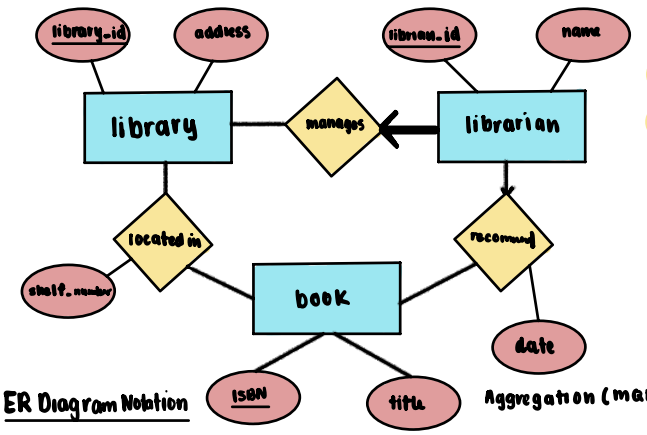
client

Project

2 columns

Entity Relationship Diagram

can = "optional"



ER Diagram Notation

Edge / Arrow	Bounds	Use Case
———	[0-many]	"not have, up to mul"
═══	[1-many]	" at least one"
——→	[0-1]	" at most one"
═══→	[1-1]	" exactly one"

Functional Dependency + Normalization

$X \to Y$ iff : $t.X = u.X \to t.Y = u.Y$

$XY \to Z$ iff : $t.X = u.X + t.Y = u.Y \to t.Z = u.Z$

Pros (+)	Cons (-)
• remove redund. • min update/ delete anomalies	• joins are costly

Normalization splitting / decomposing relations minimize redundancy

Functional dependencies constraints btw two sets of attributes

More Mongo

$all, $in, $exists, $unwind, $lookup, $insert_many, $update_many, $or, $and

```
db.table.aggregation([
  {$group: {_id: "field", newField: {$aggFunc: "$field"}}},
  {$match: {column: {$gte: 150000}}},
  {$sort: {field: -1/1}},
  {$project: {field: 0|1}},
  {$limit: {K}}
```

Aggregation (MQL): db.paintings.aggregate([
{$group: {_id: "artistId", totalPrice: {$sum: "$purchasePrice"}}},
{$match: {"totalPrice": {$gte: 100 000 000}}},
{$sort: {"totalPrice": -1}}])

Updates (MQL): db.paintings.updateMany([
{"purchasePrice": {$lt: 5 000 000}},
{$inc: {"purchasePrice": 1 000 000},
$set: {"status": "appreciated"}}])

DBMS	MDB
Database	Database
Relation	collection
Row	Document
Column	Field

Olap in PostgreSQL

year	month	sales
2022	January	100
2022	February	150
2022	March	200
2023	January	180
2023	February	120

```
SELECT year, month, SUM(sales)
FROM sales_data
GROUP BY ROLLUP(year, month);
```

```
SELECT year, month, SUM(sales)
FROM sales_data
GROUP BY CUBE (year, month);
```

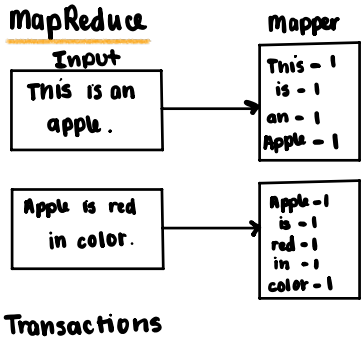
year	month	sum
2022	January	100
2022	February	150
2022	March	200
2022	NULL	450
2023	January	180
2023	February	120
2023	NULL	300
NULL	NULL	750

+		
NULL	January	280
NULL	February	270
NULL	March	200

ROLLUP - generates a result that shows aggregates for a hierarchy of values

CUBE - generates a result set that shows all possible combos of aggs for the specified group by elements

MapReduce



MongoDB

Q: list of mems for movieId 172
↳ db.movies.agg([{$unwind: "$cast"},
{$match: {movieId: {$eq: 172}}}])

Q: movies w/ rating higher than 7
↳ db.ratings.find({"ratingInfo.rating": {$gt: 7}})

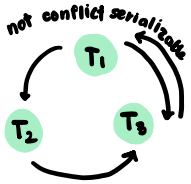
Q: Leo DiCaprio movies
↳ db.movies.find({cast: "Leo DiCaprio"})

Transactions

Time	1	2	3	4	5	6	7
T1	R(A)						W(C)
T2		W(A)		W(B)			
T3			W(A)		W(B)	W(C)	

Conflicting Actions

$R_1(A), W_2(A)$
$W_2(A), W_3(A)$
$R_1(A), W_3(A)$
$W_2(B), W_3(B)$
$W_3(C), W_1(C)$



Vocab

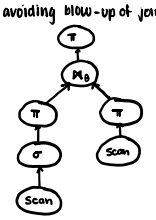
Shared (S) Lock: allow mul trans to read; prevents writing
Exclusive (X) Lock: exc. rights to read + write
Strict 2-Phase Locking: Growing (acquire) + Shrink (release)
Deadlock: when two transactions wait for held resources

Query Optimization

useful for reducing the size of relational instances and avoiding blow-up of joins

predicate pushdown: pre-select rows before join

projection pushdown: pre-select columns join



ex.)
```
SELECT c.name, S.name, S.prev_margin
FROM Candidate AS C
    INNER JOIN states AS S
        ON c.homestateID != S.sid
WHERE S.population > 6000000 AND
```

3 components of window function:

1. PARTITION BY: groups the rows so that functions will be calculated within those groups instead of the entire set of rows
2. ORDER BY: sort rows in window frame if the order of rows is important (eg. running totals)
3. RANGE: specify the window frame's relation to current row

ex.)
```
SELECT student_name, order_id, SUM(price)
OVER (PARTITION BY student_id) AS total_spent
FROM orders
ORDER BY order_id;
```

ORDER BY order_id

RANK() OVER (PARTITION BY ___ ORDER BY ___ DESC) AS _rank

Summary

Views
- ↳ virtual: output not stored, computed on demand
  - CTEs: used within larger queries
- Materialized: stored on disk, periodically refreshed

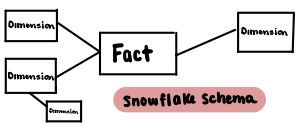
ACID Principles: properties of database transactions

Atomicity: "all or nothing" nature of transactions

Consistency: ensures valid states (ie. integrity constraints like uniqueness, PKs)

Isolation: ensures that the concurrent execution of diff transactions leaves database in same state as if transactions executed sequentially

Durability: ensures that once transaction is committed, effects → permanent even in the case of systems failure

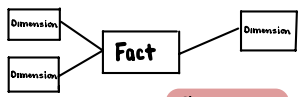


Snowflake Schema

Physical models of Data Cubes

foreign keys < primary keys, quantitative data

- still one fact table
- many dimension tables
- dimension table connected to others



Star schema

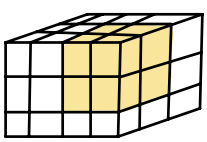
Data Cube OLAP Queries

OL(Analytical)P: reads/sums large volumes of multidimensional data

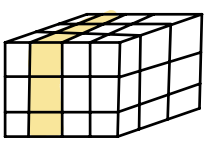
OL(Transaction)P: read, insert, update, delete data; smaller storage

↳ each element in the cube = the unique intersection of unit combo of values along dim.

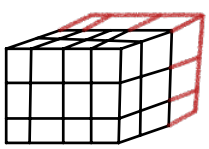
MOLAP
ROLAP
HOLAP



Dice: get range partition on one or more dimensions (subcube)

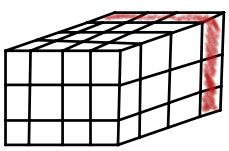


Slice: to get equality condition on a dimension



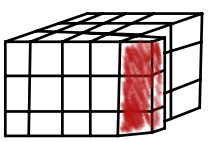
roll-up: decrease granularity to sum at higher levs of dim hierarchy

Summarize



drill-down: increase granularity to sum at low levs of a dim hierarchy

lower level



cross-tab: pivot table, dist of 1 var relative to another

OLAP systems fundamentally provide comprehensive, summarized views of data in data warehouses. Sum will happen across multiple dimensions of data...

Relational Algebra

Union: R ∪ S
Intersection: R ∩ S
Theta Join: R ⋈θ S
Natural Join: R ⋈ S

- projection (π) ex. $\pi_{id, citation}(stops)$ Push Ly
- selection (σ) ex. $\pi_{id, citation}(\sigma_{age<50}(stops))$
- renaming (ρ) ex. $\rho_{stops\_data(id \to person\_id)}(stops)$
- cartesian/cross product (×) ex. $\sigma_{stops.location = zips.location}(stops \times zips)$

DML (Data manipulation Language)

Insertion

```
INSERT INTO Relation VALUES (<list>)
```

↳ ex. `INSERT INTO stops VALUES (5000, 'Asian', 24, 'West Oakland')`

Updating

```
UPDATE Relation
SET <list of new column value assignments>
WHERE <condition>
```

↳ ex.)
```
UPDATE Stops
SET age = 18
WHERE age IS NULL;
```

Deleting

```
DELETE FROM <relationname>
DELETE FROM <relationname> WHERE <condition>
```

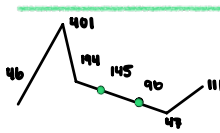
↳ ex. age < 18

Data Models

data frame – row/col labels
matrix – no labels
relations – column labels

tensors – generalization of matrix
- Rank 0, scalar
- Rank 1, vector
- Rank 2, matrix
- Rank 3, tensor

General Interpolation



194 − 49 = 145
145/3 = 49
194 − 49 = 145
145 − 49 = 96

DDL (Data Definition Language)

CREATE: defines schema
DROP: deletes both schema and instance
ALTER: redefines schema

ex.)
```
CREATE TABLE Zips(
    location VARCHAR(20)
    zipcode INTEGER
);
```

↳ ex.)
```
CREATE TABLE cast_info
    (person_id INT FOREIGN KEY REFERENCES Actor(actor_id)
```

↳ ex.) location VARCHAR(20) NOT NULL,
arrest BOOLEAN DEFAULT False,
PRIMARY KEY (stopID)
UNIQUE (personID, stopTime)

Common Table Expression EX

```
WITH relevant_authors AS (
    SELECT *
    FROM Authors
    WHERE debut_year > 2018)
SELECT DISTINCT b.genre
FROM relevant_authors AS a
INNER JOIN Books AS b ON a.aid=b.id;
```

Materialized View EX

```
CREATE MATERIALIZED VIEW _ AS
```

Subquants EX

```
SELECT name
FROM Authors
WHERE aid NOT IN (SELECT DISTINCT aid FROM Books);
```

Multidimensional OLAP	Relational OLAP
cube stored as MD array	cube stored as tables
fast responses	supports gen SQL tools
expensive ETL time	no pre-computation
pre-comp mat. slices	implemented in SQL

Spreadsheets conceptual model

A spreadsheet workbook comprises many sheets
- each sheet has cells
- a spreadsheet is structured around cells
- cells contain one or more of:
  - ↳ value
  - ↳ formula (arithmetic/special functions)

Access Control

```
GRANT [privileges] ON object TO users [WITH GRANT OPTION(user)]
REVOKE privileges ON object TO users {RESTRICT | CASCADE}
```

Functional dependencies

X Y → Z

Address	SSN	phone number
10 Green	123-456-789	(201) 233-1456
10 Green	123-456-789	(201) 123-3439
431 Purple	987-654-321	(145) 241-2131
431 Purple	987-654-321	(312) 123-1287
...	...	...

Address	SSN
10 Green	123-456-789
431 Purple	987-654-321
...	...

SSN	phone number
123-456-789	(201) 233-1456
123-456-789	(201) 123-3439
987-654-321	(145) 241-2131
987-654-321	(312) 123-1287
...	...

Update anamolies:

If we update address of person with phone number (201) 233-1456

... there will be two addresses of that person

Deletion anamolies:

If we delete the phone number for a person ... we need to check another phone number exists for said person or we lose address info (dangling pointers)

K 10^3 bytes
M 10^6
G 10^9
T 10^{12}
P 10^{15}
E 10^{18}
page = 8kb

clust. index ↑ sort time, order, min/max ↓ writing
views - memory - easy reference
mat. views - disk - query manual update (low recomp)
cte - memory - same as query (visual)
table - disk - perma data

Hierarchy - granularity lvl (time = mins, hrs, days)
numer. gran. - sig. digits, roll up → coarser (up in hier) less specific

✓ delete from child
✗ delete from parent (unless cascade)

Database = records · record-size
Pages needed = $\frac{\text{usable space per page}}{\text{bytes/record}} : \lambda \to \frac{\text{record}}{\lambda}$

df.melt = unpivot df.pivot(x, y, val)

sparse = many NaNs

Epoch - 4 byte int int 16 = 2 byte, 32 = 4 byte, 64 = 8
String - timestamp - 20 bytes double col = 8
(1 byte for each spot) page = 8 kb
SQL timestamp = 8 1 letter = 1 byte
Extract(Year from Timestamp)
TO_Timestamp(epoch) at Time Zone 'UTC' as timestamp_utc
String :: int → convert to int
int 32 = $2^{32} = 4.3 \times 10^9$ byte = 2^8 8 bits
16 = 65k 64 = 10^{19}

Nested - small table, ind on join col X big X no index (n^2)
not equi join (!=) small table
Sort-merge - medium table, data sorted/index both on join, X dup values
if input needs sort
Hash - large tables, no index, equi join X range <, >

order by random() expensive (full table)
tablesample Bernoulli(p) faster (p for each row) %
tablesample System(p) fastest (selects whole page)

DF - rows, cols have labels
mat - no labels
relation - col. labels
rank 0 - scalar (one val)
1 - vector
2 - matrix
3 - tensor (it's all)

MAD = median of deviations
Hampel = 1.4826 · MAD
2 Hampel = 2σ
MDL - takes default if possible
percen_disc(0.5) within group(order by deviation)

Mongo
.find({"key": "value", ...}) .agg
$match: {
"last": {"$gte": 37, "$lte": 41}
$group ... avg_stars: {"$avg": "$stars"}
.update_many() → '$text'
'$search': "a b c"
"$set": {"field": 'val'} field: {$exists: true} array to multiple rows
$sort
$project
$limit
$all, $in, $exists, $unwind, $lookup
$insert_many $update_many w/ $set join
xlookup(value, where, return)

$group: { _id: '$genre',
count: {$sum: 1}

Conflict = order is not good (T_2 between T_1 transactions)

T_1	RJ	WP			S	T_1			RJ	WP
T_2			RP	WP		T_2	RP	WP		

if conflict cycle = unserializable

Serial

T_1			RJ	WP	S	T_1		RJ		WP	S2
T_2	RP	WP				T_2	RP		WP		

T_1		RJ		WP		T_1		RJ	WJ		X
T_2	RP		WP			T_2	RD			WP	

not serial but is valid

conflict = diff trans, same obj, one is write
ie: T1 → R1(x) T2 → W2(x) T1 → W1(x)
DBT from {{ref('name')}}
roll up = more general day → month
$ = anchor (excel)
mult x by 8 → x = x << 3
2^3 spots shifted
cant divide

rollup(yr, mon, val) = n+1			
	2023	Jan	100
	2023	F	200
	2023	null	300

cube(yr, mon, val) = 2^n			
	2023	Jan	100
	2023	F	200
	2023	null	300
	2024	Jan	1000
	null	Jan	1200

cube math: (# unique + 1) · (# unique + 1)

serial = ordered = can do all T_i x y z

same type = M, T

Col. label = R, df

row reference by address: DF, M, Γ, S

Res. sample = SRS n big O, n rows
total not known (streams)

End. res =