SQL

Syllabus

1. Basic Queries
2. Cross Tables
3. Join
4. Aggregation
5. Data Definition
6. Data Modification
7. Null values & Subquery
8. Set operation & Sub query
9. Correlation
10. Constraints
11. Trigger

① BASIC SQL:-

SELECT : attribute
FROM : table
WHERE : predicate → Boolean Logic either 0 or 1.
;

→ *Not a
*a AND a
*a OR a

sql is case insensitive but I AM NOT!

② Cross Tables

↳ a single table does not contain desired attributes.

example :

```
SELECT   city
FROM     cityInfo , countryInfo
WHERE    cityInfo.countryid = countryInfo.country_id AND
         country = 'China';
```

$\boxed{\text{table A} \times \text{table A}}$

if same city but different address.

```
SELECT   a1.city_id
FROM     address AS a1 , address AS a2
WHERE     a1.city_id = a2.city_id AND
          a1.address_id <> a2.address_id
```

③ JOIN

Natural JOIN : considers only those pairs of tuples with same value on common attributes. No duplication.

SELECT * FROM City NATURAL JOIN country

↓

SELECT city.countyid, city.lastupdate, city_id xity county
FROM city, country
WHERE city.country_id = country.country_id
AND city.lastupdate = country.lastupdate

these are common attributes.

* NATURAL : all common attributes have same value
* ON <P> : <P> → True.
* USING (A_1, A_2) : common attributes given.
  ↳ this is best since no duplication of colums.
  → can't put any condition they must be common attributes in both table.

Join Type

<table 1> NATURAL LEFT OUTER JOIN <table 2>

if any mismatch then table 2 attribute → NULL

<table 1> NATURAL RIGHT OUTER JOIN <table 2>

table 1 ← Null table 2 ← remains.
mismatched

<table 1> NATURAL FULL OUTER JOIN <table 2>

both remains.

INNER JOIN → No unmatched tuple kept.
↳ default on JOIN

(4) Aggregations :- functions that return a single value.

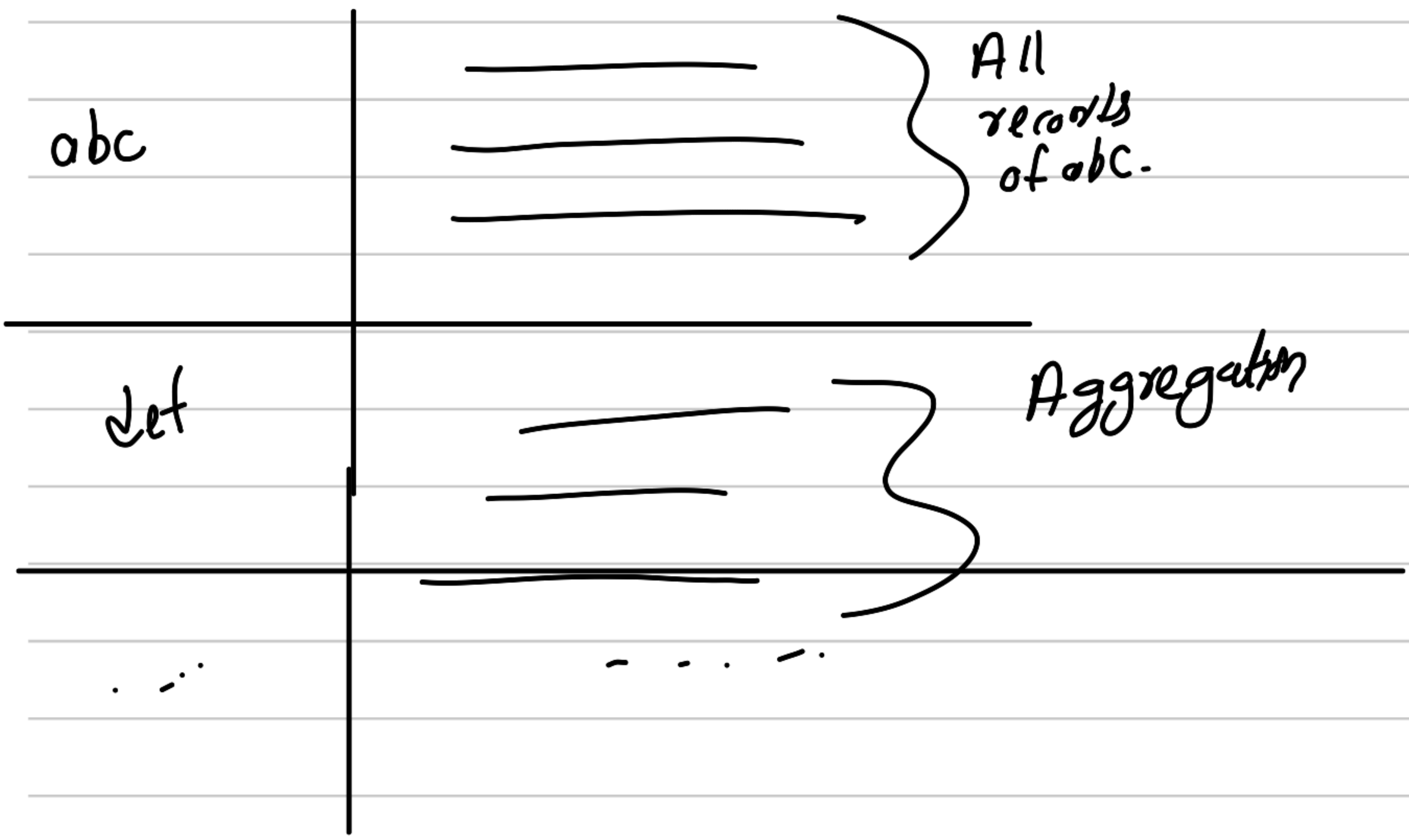
Aggregate Functions :
AVG
MIN
MAX
SUM
COUNT

SELECT
COUNT (*)
FROM Language.
↓
in order to avoid duplicates →

SELECT COUNT (DISTINCT actor_id) FROM film_actor.

GROUP BY →

Sname	max	min	avg
abc	5000	1000	2500
def			
ghi			
xyz			



Finding the number of films for each language

```
SELECT   language_id, name, COUNT(film_id)
FROM     film JOIN language USING(language_id)
GROUP    BY   language_id.
```

GROUP by attribute needs to be a key

because you are basically aggregating

actor table

actor_id	first_name
1	John
2	Emma
⋮	⋮

film_actor

actor_id	film_id
~~mst.~~ 1	101
1	102
2	101

1 → 101, 102] count = 2

HAVING → post aggregation condition.

Null → unknown value or value that does not exist.

can do:

→ calculations in expressions.

→ aggregations

→ comparisons

→ predicates.

* SELECT * FROM staff WHERE picture IS NULL

* 1 + ∅ = ∅

* all aggregations functions ignore NULL —

* Count may → 0 when all are NULL.

Logic of NULL:

TRUE, FALSE, UNKNOWN

→ T OR UNK = T
→ F OR UNK = UNK
→ UNK OR UNK = UNK

→ T AND UNK = UNK
→ F AND UNK = False
→ UNK AND UNK = UNK

NOT (UNKNOWN) = UNKNOWN

WHERE (UNKNOW):
Not run → same as false.

Sub Query :-

Nesting.

SELECT A1 ... An

FROM r1 .... rm

WHERE P

Can be replaced by subquery that generates a single value.

replaced by any valid subquery!

WHERE p → B <operation> (sub q)

Data Definition

* Create Database

```
CREATE DATABASE uic_example
```

* Create Table

```
CREATE TABLE name (
    atr_name atr type,
    atr name atr type,
    ...
    Constraint I,
    ...
)
```

char(n) ← fixed length
varchar(n) ← variable length
int
smallint
numeric(p, n) →
real
float(n)
year : 1901
date : YYYY-MM-DD
time : HH:MM:SS
blob → image
clob → long text.

Primary Key is NOT NULL.

he table for "borrow".

```
CREATE TABLE borrow(
    id            INT  NOT NULL,
    ISBN          INT  NOT NULL,
    return_date   DATE,
    borrow_date   DATE NOT NULL,
    PRIMARY KEY (id, ISBN)
)
```

← FOREIGN KEY (id) REFERENCES student (id)

* To add foreign key to existing table

ALTER TABLE borrow ADD FOREIGN KEY (ISBN) REFERENCES book(ISBN)

* Insert

INSERT INTO program (p_code, p_name, division, director_id) VALUES (1001, 'Computer Science', 'Science & Technology', NULL);

* update

UPDATE program
SET director_id = (
SELECT id
FROM instructor
WHERE i_name = 'S.T. Kwok')
WHERE p_name = 'Computer Science'

* DELETE

DELETE FROM instructor WHERE i_name = 'S.T. Kwok'

If there is an error that mean
we have to change the
programdirector to NULL first.
Then remove.

Set Ops, WHERE Clause Subqueries →

* Union * Intersection * Set Difference.	* IN subquery * NOT IN subquery.

$r = (A, B)$ $s = (A, B)$

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$

A	B
α	1
α	2
α	2
β	1
β	3

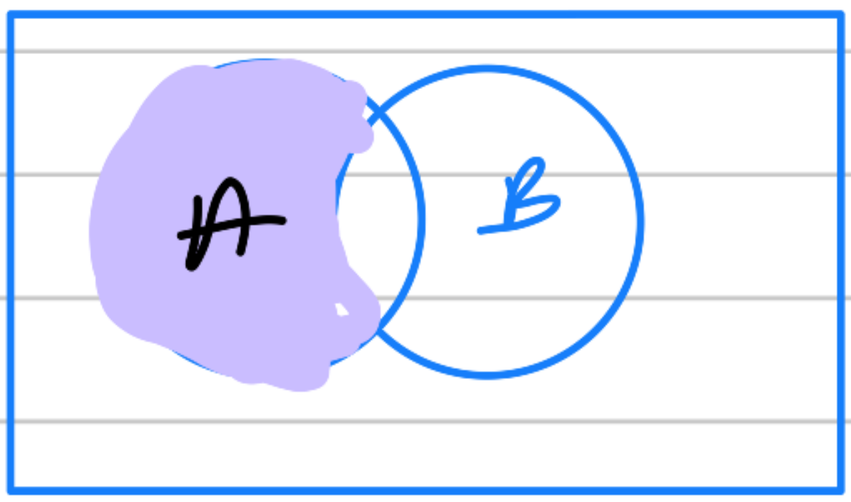
$r - s$

A	B
α	1
β	1

A − B

$r \cap s$

A	B
α	2



(Q) title of films played by actor "Bob" or "Zero"

```
( SELECT title, first_name
  FROM actor JOIN film_actor USING
    (actor_id) JOIN film USING (film_id)
  WHERE first_name = 'Bob' )
```

↓ same type.

↑ MYSQL generates duplicates.

```
U NOIN
( SELECT title, first_name
  FROM actor JOIN film_actor USING
    (actor_id) JOIN film USING (film_id)
  WHERE first_name = 'Zero' )
```

(Q) Find the id of films which are played by Tim Hackname but not in English.

```
( SELECT film-id
  FROM film-actor JOIN actor USING (actor-id)
  WHERE first_name = 'Tim' AND last-name = 'Hackman' )
EXCEPT
( SELECT film-id
  FROM film JOIN language USING (language-id)
  WHERE name = 'English' )
```

---

* Division:

Retrieve all Course names that is/are taught by all programmers!

→ first all × Noting.

→ second all ⟶ use division.

Applicants	÷	Desired skills	=	Applicants with all desired skills

Division "is like" cartesian product inverse.

catalogue ÷ programme

for each tuple t in catalogue

select all tuples t' in catalogue such that

t'.cname = t.cname

Put those t'.pname in temp

if temp covers all p-name in programme

t.cname is the result.

Catalogue

cname	pname
Database	CST
OS	CST
Database	DS
Data Mining	DS

Programme

pname
CST
DS

C ÷ P

C.name
Database

→ temp.

```
SELECT  C1.c-name
FROM    catalogue AS C1
WHERE   NOT EXISTS (
        SELECT  p-name
        FROM    programme p
        EXCEPT
```

```
(SELECT C2.phone
  FROM Catgue AS C2
  WHERE C2.cmane = C1.cmae)
```

MORE TO ADD !

book (b_id, title, YR_PUB)
user (cardNo, name, city)
borrow (b_id, cardNo, DOI)

sup (b_id, sname, prio, DOS)

(Q) Find name of that supplier who has supplied $\boxed{\text{all}}$ the books issued to abc.

b_id		sname
	← abc	x
101	→ x	
102	→ x	
103	→ x	
104	→ x	

Constraints

Integrity constraints guard against accidental damage to the database by ensuring that authorized changes to the database do not result in loss of data consistency.

⟹ * e.g: gpa must be 0.0 to 4.0

↓ in SQL

```
ALTER TABLE student
ADD CONSTRAINT gpa_range
CHECK (gpa >= 0.00 AND gpa <= 4.00)
```

⟹ No foreign key references results in NULL!

↳ if a row in foreignkey table is delete then this table needs to be updated.

```
ALTER TABLE student
ADD CONSTRAINT student_program
    FOREIGN KEY (pcode) REFERENCES program(p_code)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

→ change! on change!

on removal it will be either DEFAULT OR NULL!

TRIGGERS → statement that the system executes automatically as a side effect of a modification to the database.

```
CREATE TRIGGER <trigger-name>
BEFORE / AFTER <event>
FOR EACH ROW
< Trigger body >
```

↳ insert, update, delete.

↳ actions to be taken on satisfaction.

```
ALTER TABLE student
ADD CONSTRAINT gpa_range
CHECK (gpa >= 0.00 AND gpa <= 4.00)
```

```
CREATE TRIGGER delete_program_student
AFTER DELETE ON program
FOR EACH ROW
DELETE FROM student
WHERE student.pcode = old.pcode
```

DELIMITER:-

↳ basically don't treat ; as end!

DELEMITER { changed.

```
CREATE TRIGGER my-trigger
BEFORE INSERT ON students
FOR EACH ROW
```

```
BEGIN
        INSERT INTO logs VALUES ("Trigger")
        UPDATE stats SET count = count + 1
END;
```

DELEMETE :

additional queries.

* DELETE FROM ___ WHERE name = ';
* DROP TABLE <std>
* ALTER TABLE ....
  ADD email VARCHAR(CC);

EXISTS → will return true when result of subquery is TRUE.

```
SELECT  bid
FROM    'user u
WHERE   EXISTS
( SELECT  bid
  FROM    borrow B, supp S, usr u1
  WHERE   B.bid = S.bid AND
          u1.cardNo = B.cardNo AND
          u1.cardNo = u.cardNo AND
          smane = 'abc' )
```