



기술세미나

효율적인 SQL



우리에프아이에스

2023.06.08
고명재, 김현지,
안제현, 이소영



목차

1. 주제 선정 이유
2. 효율적인 SQL
3. 실습 예제
4. 결론



1. 주제 선정 이유



금융 IT 인재의
필요 능력



기업의 SQL
최적화 방안



데이터 조회의
중요성



SQL문에 따라
달라지는 속도



2. 효율적인 SQL

속도

메모리
관리

가독성

데이터
관리



2. 효율적인 SQL

2.1 INDEX

2.2 PARTITIONING

2.3 WINDOW FUNCTION



2.1 INDEX

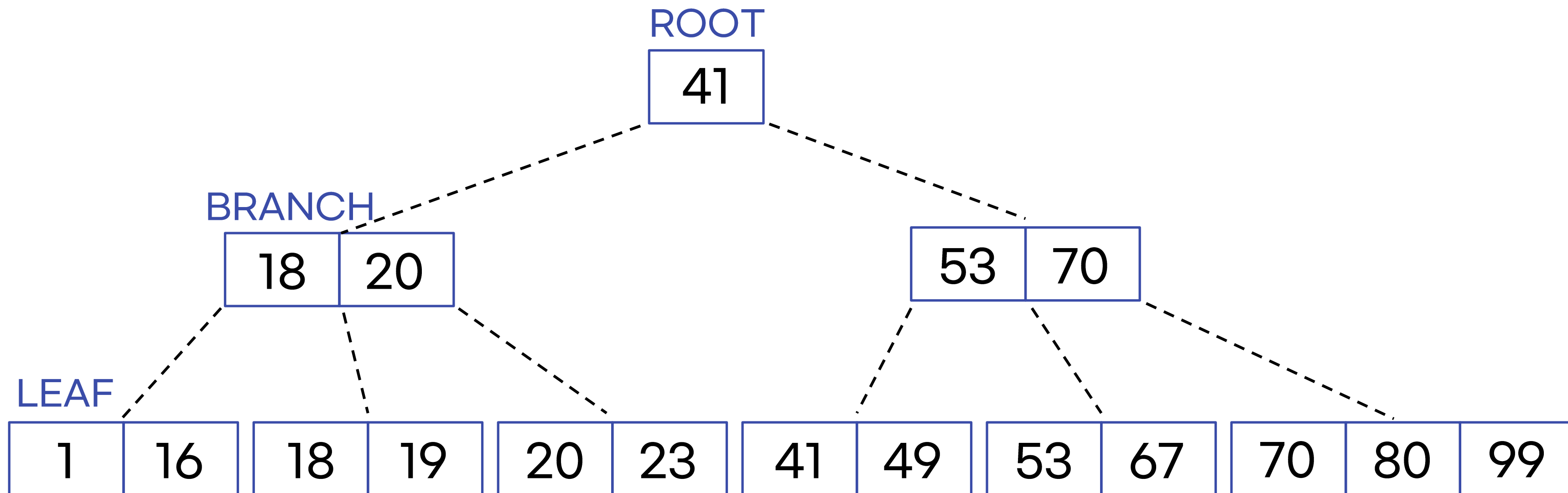
INDEX 란?

- 일정한 순서에 따라 정리
- 테이블에 대한 검색 속도를 높여주는 자료구조
 - SELECT문과 사용 시 성능 향상



2.1 INDEX

B+TREE INDEX



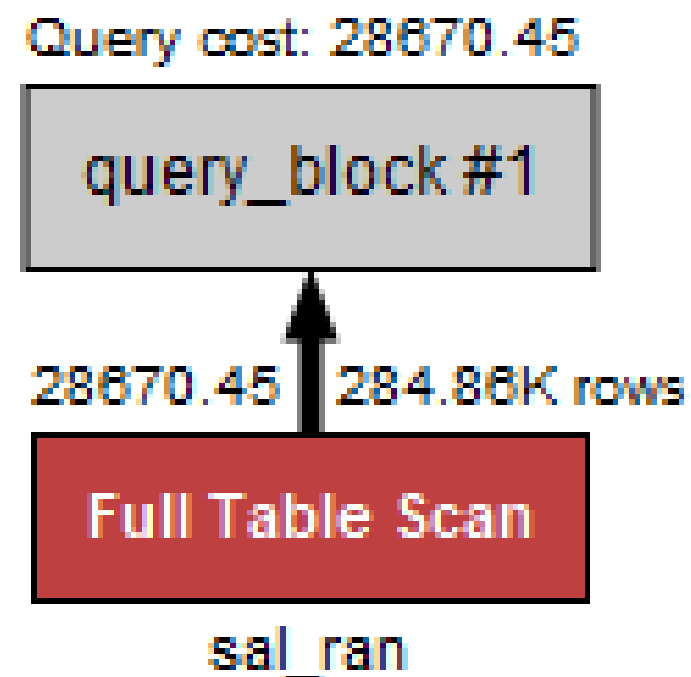
: 실제 데이터는 **LEAF NODE**에만 저장



2.1 INDEX

```
SELECT * FROM sal_ran WHERE emp_no = 20000;
```

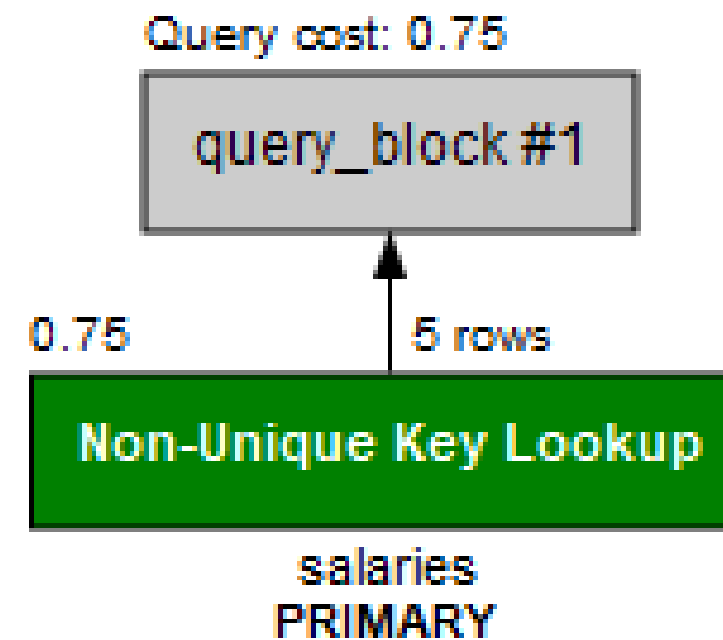
>> 실행 시간 0.187 sec



2.1 INDEX

```
SELECT * FROM salaries WHERE emp_no = 20000;
```

>> 실행 시간 0.016 sec



2.1 INDEX

SELECT

CREATE
.....

UPDATE
.....

DELETE
.....

★ 데이터 조회 속도가 중요

데이터 위치 확인,
수정 과정이 필요하여
성능 저하 가능성



2.2 PARTITIONING

PARTITIONING 이란?

- 큰 테이블을 Partition이라는 단위로 분할하는 것
- Data Partitioning은 대량의 데이터 관리에 용이



2.2 PARTITIONING

PARTITIONING 장점

- 가용성 (Availability)
전체 데이터 훼손의 가능성이 줄어들고 가용성 향상
- 관리용이성(Manageability)
DB의 관리를 쉽게 함



2.2 PARTITIONING

PARTITIONING 종류

Range partitioning

Bank Deposit Year

202101
202102
202103
...



연속적인 숫자 기준

List partitioning

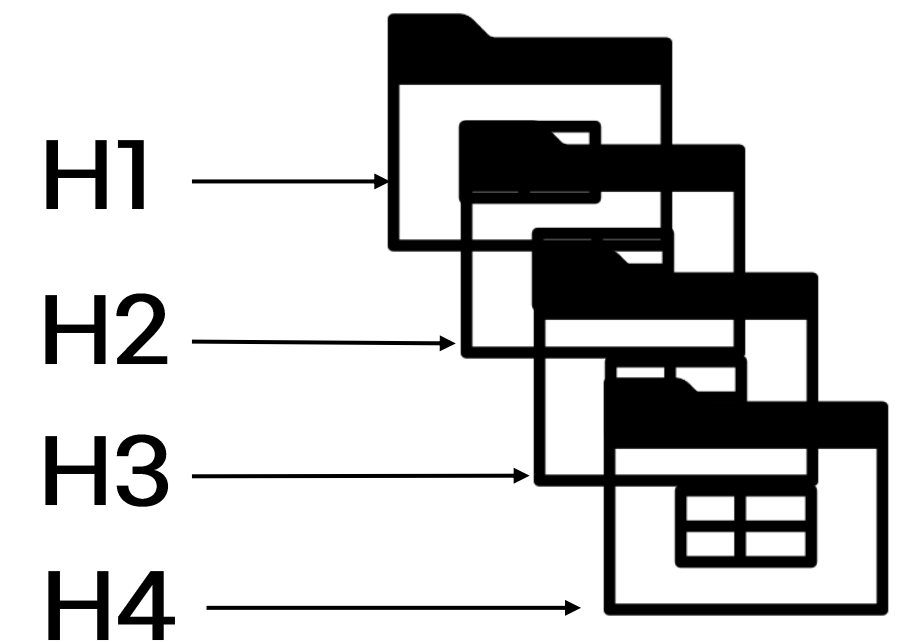
Bank Region_서울

마포구
종로구
강남구
...



명시적 제어

Hash partitioning

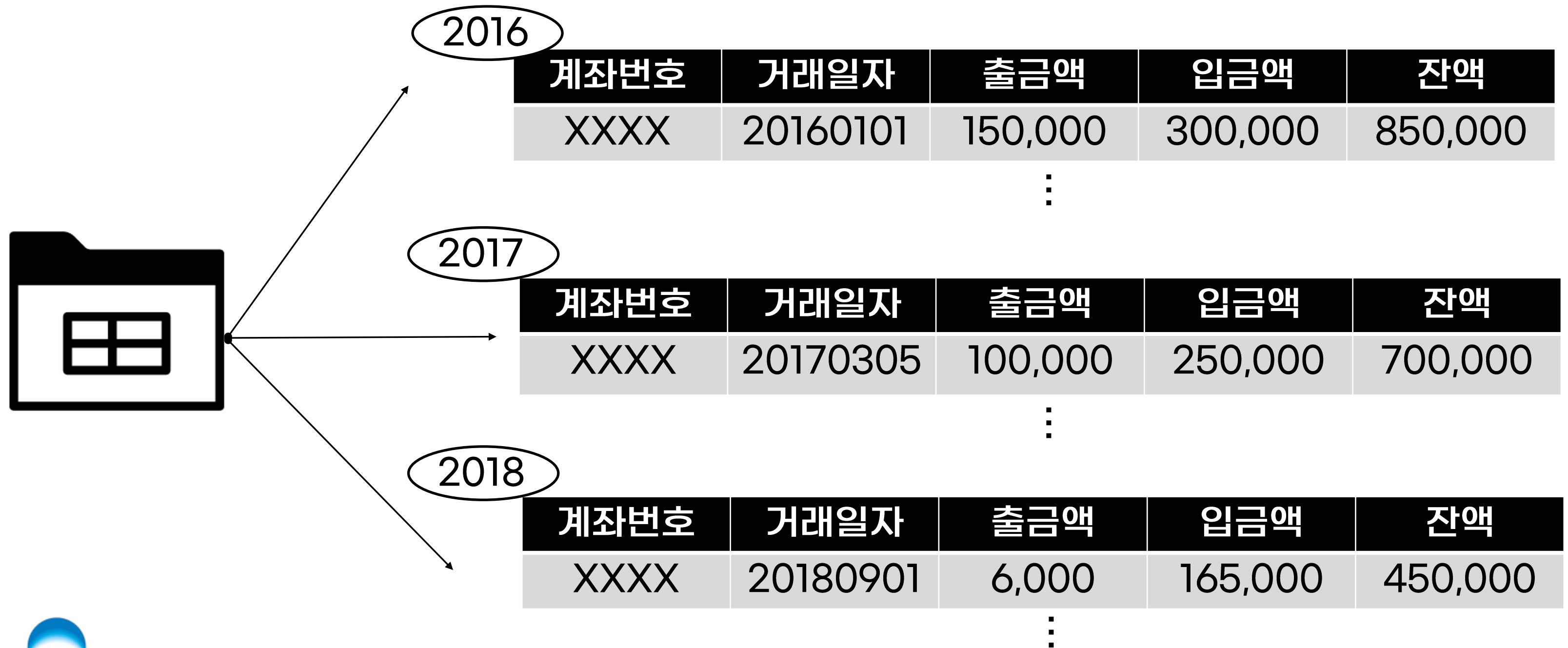


균등한 데이터 분할



2.2 PARTITIONING

Range partitioning




2.2 PARTITIONING





Partitioning Table 만들기

```
CREATE TABLE bank_year (Account_no char(30), Tran_date date,  
Pay_amt double, Deposit_amt double, Balance_amt  
double)engine=innnoDB  
partition by range(year(Tran_date)) (  
Partition y1 values less than(2016),  
partition y2 values less than(2017),  
...  
partition y5 values less than maxvalue);
```



2.2 PARTITIONING



Result Grid |   Filter Rows: | Export:  | Wrap Cell Content: 

	TABLE_SCHEMA	TABLE_NAME	PARTITION_NAME	PARTITION_ORDINAL_POSITION	TABLE_ROWS
▶	SQL_IMPROVE	bank_year	y1	1	15743
	SQL_IMPROVE	bank_year	y2	2	30248
	SQL_IMPROVE	bank_year	y3	3	28797
	SQL_IMPROVE	bank_year	y4	4	34256
	SQL_IMPROVE	bank_year	y5	5	5213






2.2 PARTITIONING

PARTITION w/o Table에서 Delete

```
DELETE FROM bank WHERE Tran_date BETWEEN  
'2017-01-01' AND '2017-12-31';
```

>> 실행 시간 0.922sec

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	DELETE	bank_nospace_sample	NULL	ALL	NULL	NULL	NULL	NULL	87052	100.00	Using where







2.2 PARTITIONING

PARTITION Table에서 Delete

```
DELETE FROM bank_year WHERE Tran_date BETWEEN  
'2017-01-01' AND '2017-12-31';
```

>> 실행 시간 0.578sec

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	DELETE	bank_year_sample	y3	ALL	NULL	NULL	NULL	NULL	1	100.00	Using where



2.2 PARTITIONING

PARTITION Table Drop

```
ALTER TABLE bank_year DROP Partition y3;
```

>> 실행 시간 0.125sec



2.2 PARTITIONING

관리의 효율성

- 대량의 데이터를 보유한 테이블
- 데이터의 변화가 필요한 테이블
- 특정 기준으로 데이터 정렬 및 참조가 필요한 테이블



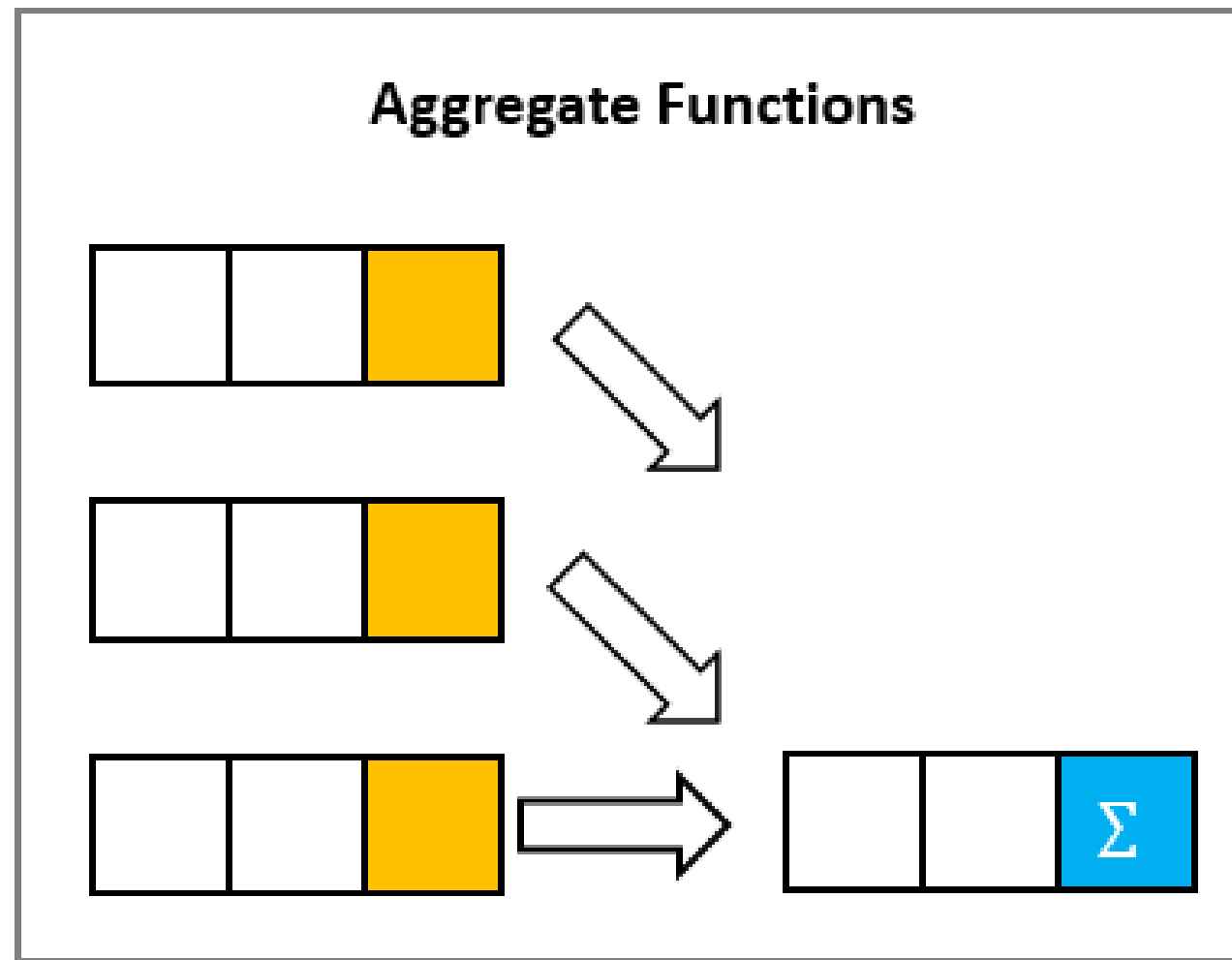
2.3 WINDOW FUNCTION

WINDOW FUNCTION 이란?

- 특정 열을 기준으로 그룹(group) 된 행(윈도우)을 연산하는 함수



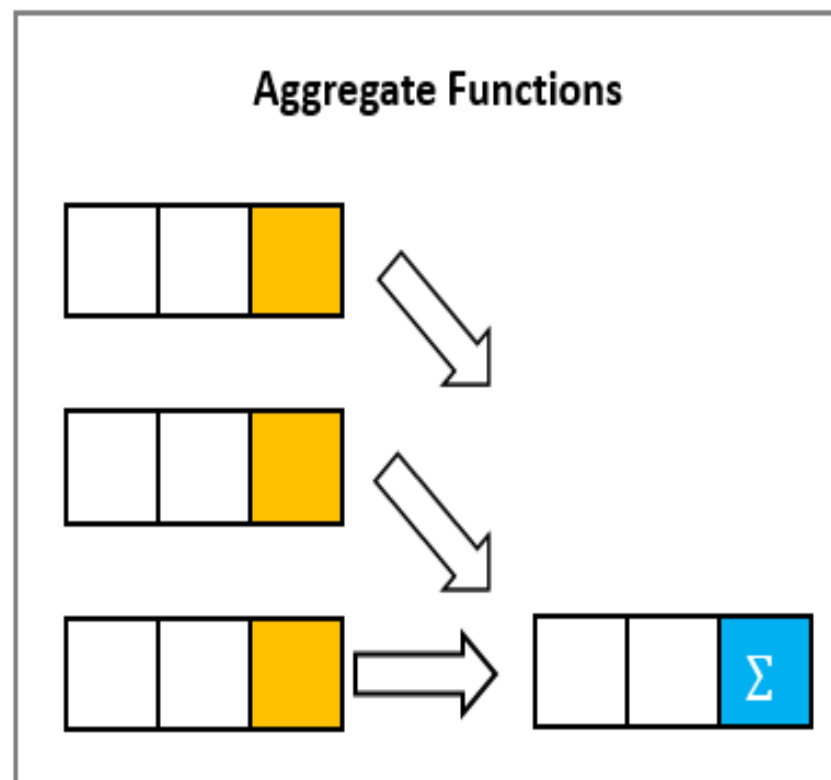
2.3 WINDOW FUNCTION



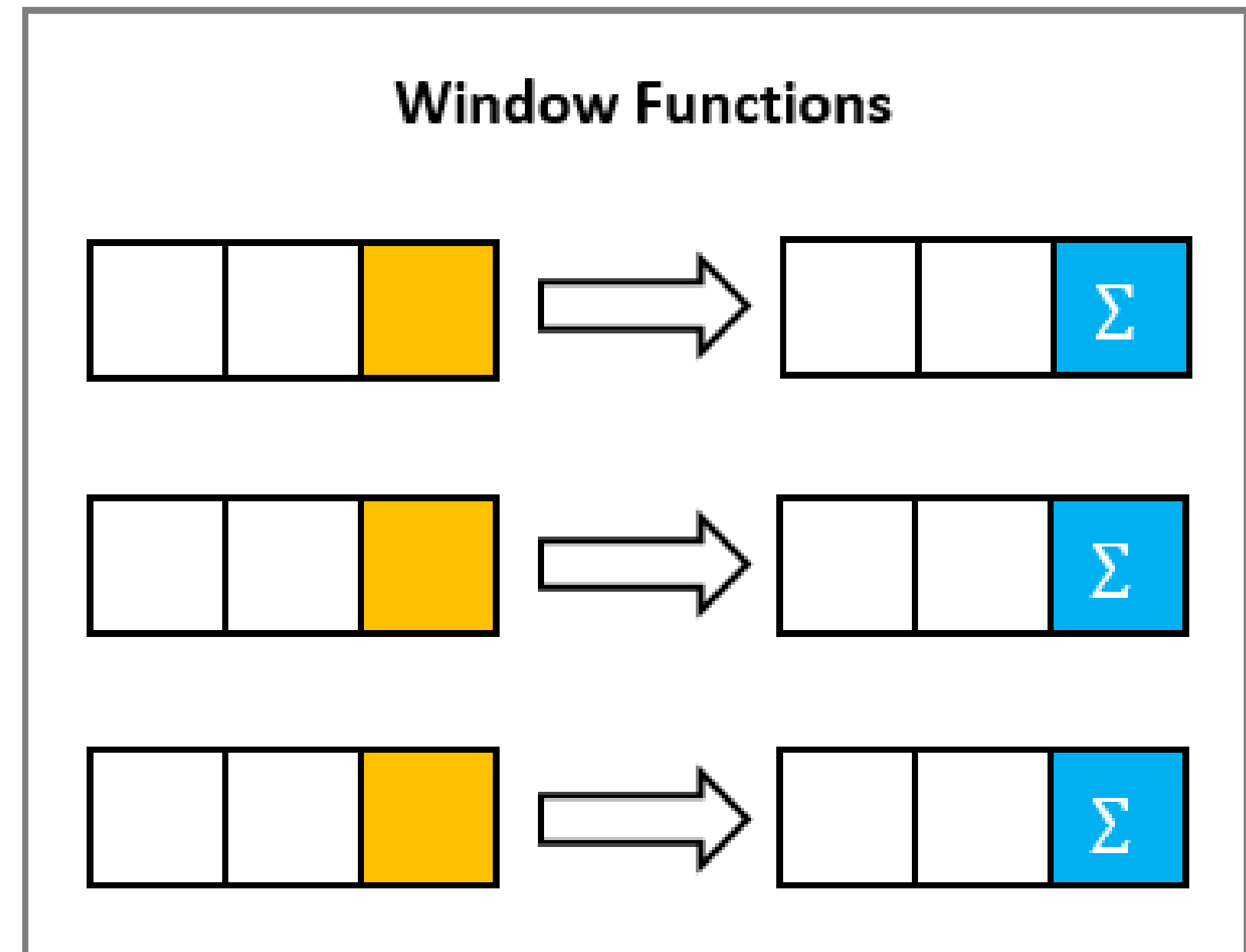
결과가 반드시 **한 행에만 출력**



2.3 WINDOW FUNCTION



결과가 반드시 **한 행에만** 출력



계산한 값을 **행마다** 출력



2.3 WINDOW FUNCTION

WINDOW FUNCTION 종류

순위 관련	RANK, DENSE_RANK, ROW_NUMBER
집계 관련	SUM, MAX, MIN, AVG, COUNT
행 순서 관련	FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 관련	CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT



2.3 WINDOW FUNCTION

WINDOW FUNCTION 종류

순위 관련	RANK, DENSE_RANK, ROW_NUMBER
집계 관련	SUM, MAX, MIN, AVG, COUNT
행 순서 관련	FIRST_VALUE, LAST_VALUE, LAG, LEAD
비율 관련	CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT



2.3 WINDOW FUNCTION

WINDOW FUNCTION 구문

```
SELECT WINDOW_FUNCTION (ARGUMENTS)  
OVER([PARTITION BY 컬럼] [ORDER BY 컬럼]  
[WINDOWING 절])  
FROM 테이블명;
```



2.3 WINDOW FUNCTION

NTILE()

ex)

salaries 테이블에서

salary(급여)가 높은 순서대로 1~4등급으로 분류하기



```
SELECT COUNT(*) INTO @myCount FROM salaries;
SELECT s.emp_po, s.salary, s.from_date, c.rn,
CASE
    WHEN rn >= @ myCount * 0.75
    THEN 1
    WHEN rn >= @myCount * 0.5
    THEN 2
    WHEN rn >= @myCount * 0.25
    THEN 3
    ELSE 4
END AS QUAL_TILE
FROM salaries s, (SELECT emp_no, salary, from_date,
to_date, (SELECT COUNT(*)
FROM salaries a
WHERE a.salary <= b.salary) as rn
FROM salaries b) c
WHERE s.emp_no = c.emp_no AND s. from_date = c.from_date;
```



2.3 WINDOW FUNCTION

NTILE()

```
SELECT emp_no, salary,  
       NTILE(4) OVER(ORDER BY salary DESC) QUAR_TILE  
FROM salaries;
```

>> 실행 시간 0.532sec



2.3 WINDOW FUNCTION

LAG()

ex)

salaries 테이블에서

현재 salary(급여)와 작년 salary를 비교하여 차이 구하기



```
CREATE TABLE TEMP
SELECT emp_no, from_date, salary,
COALESCE((
    SELECT salary FROM salaries s2
    WHERE s2.emp_no = s1.emp_no AND s2.from_date < s1.from_date
    ORDER BY s2.from_date DESC
    LIMIT 1
), 0) as last_year_salary,
Salary - COALESCE((
    SELECT salary FROM salaries s2
    WHERE s2.emp_no = s1.emp_no AND s2.from_date < s1.from_date
    ORDER BY s2.from_date DESC
    LIMIT 1
), 0) as 연봉차이
FROM salaries s1;
```



2.3 WINDOW FUNCTION

LAG()

```
SELECT emp_no, from_date, salary,  
LAG(salary, 1) OVER(PARTITION BY emp_no ORDER BY  
emp_no) as last_year_salary,  
salary LAG(salary, 1) OVER(PARTITION BY emp_no ORDER  
BY emp_no) as 연봉차이  
FROM salaries;
```

>> 실행 시간 1.032sec



2.3 WINDOW FUNCTION

- 테이블 스캔 측면에서 성능이 우수 -> 실행시간 감소
- 간단하고 이해하기 쉬운 쿼리 작성 -> 가독성 향상



3. 실습 예제

Application 구현



4. 결론

“어떠한 좋은 SQL문도
좋은 DB 설계를 이길 수 없다”



출처

양바른, 『업무에 바로 쓰는 SQL 튜닝』, 한빛미디어, 2021
우재남, 『이것이 MySQL이다』, 한빛미디어, 2020
홍형경, 『Let's Get IT SQL 프로그래밍』, 길벗, 2021

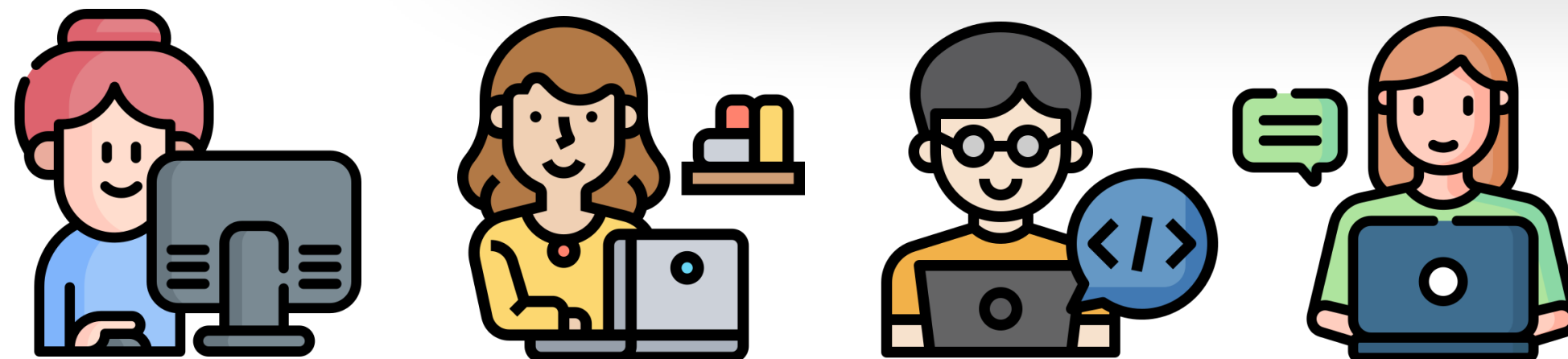
<https://www.kaggle.com/datasets/apoorvwatsky/bank-transaction-data?resource=download>

<https://www.kaggle.com/datasets/kidoen/bank-customers-data>

<https://schatz37.tistory.com/12>

<https://velog.io/@jiffydev>

<https://www.youtube.com/watch?v=edpYzFgHbqs>



Q & A





기술세미나

효율적인 SQL

경청해주셔서 감사합니다.



우리에프아이에스