ECE30030/ITP30010 Database Systems

Advanced SQL

Reading: Chapters 4-5

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Weeks #13-16 Schedule

Weeks	Topics	Notes
13	Transaction	
14	Indexes	
15	Stored procedures, Functions, Triggers	
16	Final exam (Thursday, Friday)	

Agenda

- Window functions
- Transactions

Window Functions in SQL

- Syntax
  - **SELECT WINDOW_FUNCTION** ([**ALL**] expression)
    OVER ([**PARTITION BY** partition_list] [**ORDER BY** order_list])
    FROM table;

    - **WINDOW_FUNCTION**: Specify the name of the window function
    - **ALL** (optional): When you will include ALL it will count all values including duplicates
      - *C.f.*, DISTINCT is not supported in window functions
    - **OVER**: Specifies the window clauses for aggregate functions
      - **PARTITION BY** partition_list: Defines the window (set of rows on which window function operates) for window functions
        - If **PARTITION BY** is not specified, grouping will be done on entire table and values will be aggregated accordingly
      - **ORDER BY** order_list: Sorts the rows within each partition
        - If **ORDER BY** is not specified, ORDER BY uses the entire table

Window Functions in SQL

- Window function types
  - Aggregate window functions
    - **SUM**(), **MAX**(), **MIN**(), **AVG**(), **COUNT**(), …
  - Ranking window functions
    - **RANK**(), **DENSE_RANK**(), **PERCENT_RANK**(), **ROW_NUMBER**(), **NTILE**()
  - Value window functions
    - **LAG**(), **LEAD**(), **FIRST_VALUE**(), **LAST_VALUE**(), **CUME_DIST**(), **NTH_VALUE**()

Aggregation Examples

- Average over each job
  - **SELECT** ENAME, SAL, JOB,
    AVG(SAL) **OVER** (**PARTITION BY** JOB) **AS** AVG_SAL_JOB
    FROM EMP;

ENAME	SAL	JOB	AVG_SAL_JOB
FORD	3000.00	ANALYST	3000.000000
SCOTT	3000.00	ANALYST	3000.000000
JAMES	950.00	CLERK	1037.500000
SMITH	800.00	CLERK	1037.500000
ADAMS	1100.00	CLERK	1037.500000
MILLER	1300.00	CLERK	1037.500000
BLAKE	2850.00	MANAGER	2758.333333
CLARK	2450.00	MANAGER	2758.333333
JONES	2975.00	MANAGER	2758.333333
KING	5000.00	PRESIDENT	5000.000000
MARTIN	1250.00	SALESMAN	1400.000000
ALLEN	1600.00	SALESMAN	1400.000000
TURNER	1500.00	SALESMAN	1400.000000
WARD	1250.00	SALESMAN	1400.000000

Aggregation Examples

- *C.f.*, Aggregation over groups
  - **SELECT** JOB, **AVG**(SAL)
    FROM EMP
    GROUP BY JOB;

JOB	AVG(SAL)
PRESIDENT	5000.000000
MANAGER	2758.333333
SALESMAN	1400.000000
CLERK	1037.500000
ANALYST	3000.000000

Ranking Example

- A table with the total rank and partitioned rank:
  - **SELECT** ENAME, SAL,
    RANK() OVER (**ORDER BY** SAL **DESC**) ALL_RANK,
    DENSE_RANK() OVER (**PARTITION BY** JOB **ORDER BY** SAL **DESC**) JOB_RANK
    FROM EMP;

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-05-09	2450.00	NULL	10
7566	JONES	MANAGER	7839	1981-04-01	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-10	1250.00	1400.00	30
7499	ALLEN	SALESMAN	7698	1981-02-11	1600.00	300.00	30
7844	TURNER	SALESMAN	7698	1981-08-21	1500.00	0.00	30
7900	JAMES	CLERK	7698	1981-12-11	950.00	NULL	30
7521	WARD	SALESMAN	7698	1981-02-23	1250.00	500.00	30
7902	FORD	ANALYST	7566	1981-12-11	3000.00	NULL	20
7369	SMITH	CLERK	7902	1980-12-09	800.00	NULL	20
7788	SCOTT	ANALYST	7566	1982-12-22	3000.00	NULL	20
7876	ADAMS	CLERK	7788	1983-01-15	1100.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-11	1300.00	NULL	10

Result

ENAME	SAL	ALL_RANK	JOB_RANK
FORD	3000.00	2	1
SCOTT	3000.00	2	1
MILLER	1300.00	9	1
ADAMS	1100.00	12	2
JAMES	950.00	13	3
SMITH	800.00	14	4
JONES	2975.00	4	1
BLAKE	2850.00	5	2
CLARK	2450.00	6	3
KING	5000.00	1	1
ALLEN	1600.00	7	1
TURNER	1500.00	8	2
MARTIN	1250.00	10	3
WARD	1250.00	10	3

Ranking Example

- A table with the total rank and partitioned rank:
  - **SELECT ROW_NUMBER() OVER (ORDER BY** SAL **DESC**) ROW_NUM,
    ENAME, SAL,
    RANK() **OVER (ORDER BY** SAL **DESC**) ALL_RANK
    FROM EMP;

ROW_NUM	ENAME	SAL	ALL_RANK
1	KING	5000.00	1
2	FORD	3000.00	2
3	SCOTT	3000.00	2
4	JONES	2975.00	4
5	BLAKE	2850.00	5
6	CLARK	2450.00	6
7	ALLEN	1600.00	7
8	TURNER	1500.00	8
9	MILLER	1300.00	9
10	MARTIN	1250.00	10
11	WARD	1250.00	10
12	ADAMS	1100.00	12
13	JAMES	950.00	13
14	SMITH	800.00	14

Running Examples

- DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-05-09	2450.00	NULL	10
7566	JONES	MANAGER	7839	1981-04-01	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-10	1250.00	1400.00	30
7499	ALLEN	SALESMAN	7698	1981-02-11	1600.00	300.00	30
7844	TURNER	SALESMAN	7698	1981-08-21	1500.00	0.00	30
7900	JAMES	CLERK	7698	1981-12-11	950.00	NULL	30
7521	WARD	SALESMAN	7698	1981-02-23	1250.00	500.00	30
7902	FORD	ANALYST	7566	1981-12-11	3000.00	NULL	20
7369	SMITH	CLERK	7902	1980-12-09	800.00	NULL	20
7788	SCOTT	ANALYST	7566	1982-12-22	3000.00	NULL	20
7876	ADAMS	CLERK	7788	1983-01-15	1100.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-11	1300.00	NULL	10

Ranking Examples

- Rank by salary
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    ROW_NUMBER() **OVER** (**ORDER BY** SAL) **AS** ROW_NUMBER_SAL,
    RANK() **OVER** (**ORDER BY** SAL) **AS** RANK_SAL,
    DENSE_RANK() **OVER** (**ORDER BY** SAL) **AS** DENSE_RANK_SAL
    FROM EMP;

ENAME	SAL	JOB	HIREDATE	ROW_NUMBER_SAL	RANK_SAL	DENSE_RANK_SAL
SMITH	800.00	CLERK	1980-12-09	1	1	1
JAMES	950.00	CLERK	1981-12-11	2	2	2
ADAMS	1100.00	CLERK	1983-01-15	3	3	3
MARTIN	1250.00	SALESMAN	1981-09-10	4	4	4
WARD	1250.00	SALESMAN	1981-02-23	5	4	4
MILLER	1300.00	CLERK	1982-01-11	6	6	5
TURNER	1500.00	SALESMAN	1981-08-21	7	7	6
ALLEN	1600.00	SALESMAN	1981-02-11	8	8	7
CLARK	2450.00	MANAGER	1981-05-09	9	9	8
BLAKE	2850.00	MANAGER	1981-05-01	10	10	9
JONES	2975.00	MANAGER	1981-04-01	11	11	10
FORD	3000.00	ANALYST	1981-12-11	12	12	11
SCOTT	3000.00	ANALYST	1982-12-22	13	12	11
KING	5000.00	PRESIDENT	1981-11-17	14	14	12

Ranking Examples

- Rank by hiredate
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    ROW_NUMBER() **OVER** (**ORDER BY** HIREDATE) **AS** ROW_NUMBER_HIREDATE,
    RANK() **OVER** (**ORDER BY** HIREDATE) **AS** RANK_HIREDATE,
    DENSE_RANK() **OVER** (**ORDER BY** HIREDATE) **AS** DENSE_RANK_HIREDATE
    FROM EMP;

ENAME	SAL	JOB	HIREDATE	ROW_NUMBER_HIREDATE	RANK_HIREDATE	DENSE_RANK_HIREDATE
SMITH	800.00	CLERK	1980-12-09	1	1	1
ALLEN	1600.00	SALESMAN	1981-02-11	2	2	2
WARD	1250.00	SALESMAN	1981-02-23	3	3	3
JONES	2975.00	MANAGER	1981-04-01	4	4	4
BLAKE	2850.00	MANAGER	1981-05-01	5	5	5
CLARK	2450.00	MANAGER	1981-05-09	6	6	6
TURNER	1500.00	SALESMAN	1981-08-21	7	7	7
MARTIN	1250.00	SALESMAN	1981-09-10	8	8	8
KING	5000.00	PRESIDENT	1981-11-17	9	9	9
JAMES	950.00	CLERK	1981-12-11	10	10	10
FORD	3000.00	ANALYST	1981-12-11	11	10	10
MILLER	1300.00	CLERK	1982-01-11	12	12	11
SCOTT	3000.00	ANALYST	1982-12-22	13	13	12
ADAMS	1100.00	CLERK	1983-01-15	14	14	13

Ranking Examples

- Rank by hiredate within each job
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    RANK() **OVER** (**PARTITION BY** JOB **ORDER BY** HIREDATE **DESC**) **AS** RANK_HIREDATE
    FROM EMP;

ENAME	SAL	JOB	HIREDATE	RANK_HIREDATE
SCOTT	3000.00	ANALYST	1982-12-22	1
FORD	3000.00	ANALYST	1981-12-11	2
ADAMS	1100.00	CLERK	1983-01-15	1
MILLER	1300.00	CLERK	1982-01-11	2
JAMES	950.00	CLERK	1981-12-11	3
SMITH	800.00	CLERK	1980-12-09	4
CLARK	2450.00	MANAGER	1981-05-09	1
BLAKE	2850.00	MANAGER	1981-05-01	2
JONES	2975.00	MANAGER	1981-04-01	3
KING	5000.00	PRESIDENT	1981-11-17	1
MARTIN	1250.00	SALESMAN	1981-09-10	1
TURNER	1500.00	SALESMAN	1981-08-21	2
WARD	1250.00	SALESMAN	1981-02-23	3
ALLEN	1600.00	SALESMAN	1981-02-11	4

Ranking Examples

- Rank by hiredate within each job
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    RANK() **OVER** w **AS** RANK_HIREDATE
    FROM EMP
    WINDOW w **AS** (**PARTITION BY** JOB **ORDER BY** HIREDATE **DESC**);

ENAME	SAL	JOB	HIREDATE	RANK_HIREDATE
SCOTT	3000.00	ANALYST	1982-12-22	1
FORD	3000.00	ANALYST	1981-12-11	2
ADAMS	1100.00	CLERK	1983-01-15	1
MILLER	1300.00	CLERK	1982-01-11	2
JAMES	950.00	CLERK	1981-12-11	3
SMITH	800.00	CLERK	1980-12-09	4
CLARK	2450.00	MANAGER	1981-05-09	1
BLAKE	2850.00	MANAGER	1981-05-01	2
JONES	2975.00	MANAGER	1981-04-01	3
KING	5000.00	PRESIDENT	1981-11-17	1
MARTIN	1250.00	SALESMAN	1981-09-10	1
TURNER	1500.00	SALESMAN	1981-08-21	2
WARD	1250.00	SALESMAN	1981-02-23	3
ALLEN	1600.00	SALESMAN	1981-02-11	4

Ranking Examples

- Percentile by salary within each job
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    RANK() **OVER** (**ORDER BY** SAL) **AS** RANK_SAL,
    CUME_DIST() **OVER** (**ORDER BY** SAL) **AS** CUME_DIST_SAL,
    PERCENT_RANK() **OVER** (**ORDER BY** SAL) **AS** PERCENT_RANK_SAL
    FROM EMP;

ENAME	SAL	JOB	HIREDATE	RANK_SAL	CUME_DIST_SAL	PERCENT_RANK_SAL
SMITH	800.00	CLERK	1980-12-09	1	0.07142857142857142	0
JAMES	950.00	CLERK	1981-12-11	2	0.14285714285714285	0.07692307692307693
ADAMS	1100.00	CLERK	1983-01-15	3	0.21428571428571427	0.15384615384615385
MARTIN	1250.00	SALESMAN	1981-09-10	4	0.35714285714285715	0.23076923076923078
WARD	1250.00	SALESMAN	1981-02-23	4	0.35714285714285715	0.23076923076923078
MILLER	1300.00	CLERK	1982-01-11	6	0.42857142857142855	0.38461538461538464
TURNER	1500.00	SALESMAN	1981-08-21	7	0.5	0.46153846153846156
ALLEN	1600.00	SALESMAN	1981-02-11	8	0.5714285714285714	0.5384615384615384
CLARK	2450.00	MANAGER	1981-05-09	9	0.6428571428571429	0.6153846153846154
BLAKE	2850.00	MANAGER	1981-05-01	10	0.7142857142857143	0.6923076923076923
JONES	2975.00	MANAGER	1981-04-01	11	0.7857142857142857	0.7692307692307693
FORD	3000.00	ANALYST	1981-12-11	12	0.9285714285714286	0.8461538461538461
SCOTT	3000.00	ANALYST	1982-12-22	12	0.9285714285714286	0.8461538461538461
KING	5000.00	PRESIDENT	1981-11-17	14	1	1

Ranking Examples

- Percentile by salary within each job
  - **SELECT** ENAME, SAL, JOB, HIREDATE,
    RANK() **OVER** w **AS** RANK_SAL,
    CUME_DIST() **OVER** w **AS** CUME_DIST_SAL,
    PERCENT_RANK() **OVER** w **AS** PERCENT_RANK_SAL
    FROM EMP
    WINDOW w **AS** (**ORDER BY** SAL);

ENAME	SAL	JOB	HIREDATE	RANK_SAL	CUME_DIST_SAL	PERCENT_RANK_SAL
SMITH	800.00	CLERK	1980-12-09	1	0.07142857142857142	0
JAMES	950.00	CLERK	1981-12-11	2	0.14285714285714285	0.07692307692307693
ADAMS	1100.00	CLERK	1983-01-15	3	0.21428571428571427	0.15384615384615385
MARTIN	1250.00	SALESMAN	1981-09-10	4	0.35714285714285715	0.23076923076923078
WARD	1250.00	SALESMAN	1981-02-23	4	0.35714285714285715	0.23076923076923078
MILLER	1300.00	CLERK	1982-01-11	6	0.42857142857142855	0.38461538461538464
TURNER	1500.00	SALESMAN	1981-08-21	7	0.5	0.46153846153846156
ALLEN	1600.00	SALESMAN	1981-02-11	8	0.5714285714285714	0.5384615384615384
CLARK	2450.00	MANAGER	1981-05-09	9	0.6428571428571429	0.6153846153846154
BLAKE	2850.00	MANAGER	1981-05-01	10	0.7142857142857143	0.6923076923076923
JONES	2975.00	MANAGER	1981-04-01	11	0.7857142857142857	0.7692307692307693
FORD	3000.00	ANALYST	1981-12-11	12	0.9285714285714286	0.8461538461538461
SCOTT	3000.00	ANALYST	1982-12-22	12	0.9285714285714286	0.8461538461538461
KING	5000.00	PRESIDENT	1981-11-17	14	1	1

Running Examples

- Orders

ID	ORD_DATE	CUSTOMER_NAME	CITY	ORD_AMT
1001	2017-04-01	David Smith	GuildFord	10000.00
1002	2017-04-02	David Jones	Arlington	20000.00
1003	2017-04-03	John Smith	Shalford	5000.00
1004	2017-04-04	Michael Smith	GuildFord	15000.00
1005	2017-04-05	David Williams	Shalford	7000.00
1006	2017-04-06	Paum Smith	GuildFord	25000.00
1007	2017-04-10	Andrew Smith	Arlington	15000.00
1008	2017-04-11	David Brown	Arlington	2000.00
1009	2017-04-20	Robert Smith	Shalford	1000.00
1010	2017-04-25	Peter Smith	GuildFord	500.00

Running Examples

- You can DIY…

```
CREATE TABLE ORDERS
(
	ID INT,
	ORD_DATE DATE,
	CUSTOMER_NAME VARCHAR(250),
	CITY VARCHAR(100),
	ORD_AMT DECIMAL(9,2)
);

INSERT INTO ORDERS(ID, ORD_DATE, CUSTOMER_NAME, CITY, ORD_AMT)
SELECT '1001','2017-04-01','David Smith','GuildFord',10000
UNION ALL
SELECT '1002','2017-04-02','David Jones','Arlington',20000
UNION ALL
SELECT '1003','2017-04-03','John Smith','Shalford',5000
UNION ALL
SELECT '1004','2017-04-04','Michael Smith','GuildFord',15000
UNION ALL
SELECT '1005','2017-04-05','David Williams','Shalford',7000
UNION ALL
SELECT '1006','2017-04-06','Paum Smith','GuildFord',25000
UNION ALL
SELECT '1007','2017-04-10','Andrew Smith','Arlington',15000
UNION ALL
SELECT '1008','2017-04-11','David Brown','Arlington',2000
UNION ALL
SELECT '1009','2017-04-20','Robert Smith','Shalford',1000
UNION ALL
SELECT '1010','2017-04-25','Peter Smith','GuildFord',500;
```

Value Window Examples

- First and last records in each partition
  - **SELECT** ID, CITY, ORD_DATE,
    FIRST_VALUE(ORD_DATE) **OVER**(**PARTITION BY** CITY) **AS** FIRST_VAL,
    LAST_VALUE(ORD_DATE) **OVER**(**PARTITION BY** CITY) **AS** LAST_VAL
    FROM ORDERS;

ID	CITY	ORD_DATE	FIRST_VAL	LAST_VAL
1002	Arlington	2017-04-02	2017-04-02	2017-04-11
1007	Arlington	2017-04-10	2017-04-02	2017-04-11
1008	Arlington	2017-04-11	2017-04-02	2017-04-11
1001	GuildFord	2017-04-01	2017-04-01	2017-04-25
1004	GuildFord	2017-04-04	2017-04-01	2017-04-25
1006	GuildFord	2017-04-06	2017-04-01	2017-04-25
1010	GuildFord	2017-04-25	2017-04-01	2017-04-25
1003	Shalford	2017-04-03	2017-04-03	2017-04-20
1005	Shalford	2017-04-05	2017-04-03	2017-04-20
1009	Shalford	2017-04-20	2017-04-03	2017-04-20

Value Window Examples

- First and last records in each partition
  - **SELECT** ID, CUSTOMER_NAME, CITY, ORD_AMT, ORD_DATE,
    LAG(ORD_DATE,1) **OVER**(**ORDER BY** ORD_DATE) **AS** PREV_ORD_DAT,
    LEAD(ORD_DATE,1) **OVER**(**ORDER BY** ORD_DATE) **AS** NEXT_ORD_DAT
    FROM ORDERS;

ID	CUSTOMER_NAME	CITY	ORD_AMT	ORD_DATE	PREV_ORD_DAT	NEXT_ORD_DAT
1001	David Smith	GuildFord	10000.00	2017-04-01	NULL	2017-04-02
1002	David Jones	Arlington	20000.00	2017-04-02	2017-04-01	2017-04-03
1003	John Smith	Shalford	5000.00	2017-04-03	2017-04-02	2017-04-04
1004	Michael Smith	GuildFord	15000.00	2017-04-04	2017-04-03	2017-04-05
1005	David Williams	Shalford	7000.00	2017-04-05	2017-04-04	2017-04-06
1006	Paum Smith	GuildFord	25000.00	2017-04-06	2017-04-05	2017-04-10
1007	Andrew Smith	Arlington	15000.00	2017-04-10	2017-04-06	2017-04-11
1008	David Brown	Arlington	2000.00	2017-04-11	2017-04-10	2017-04-20
1009	Robert Smith	Shalford	1000.00	2017-04-20	2017-04-11	2017-04-25
1010	Peter Smith	GuildFord	500.00	2017-04-25	2017-04-20	NULL

Value Window Examples

- First and last records in each partition
  - **SELECT** ID, CUSTOMER_NAME, CITY, ORD_AMT, ORD_DATE,
    LAG(ORD_DATE,2) **OVER**(**ORDER BY** ORD_DATE) **AS** PREV_ORD_DAT,
    LEAD(ORD_DATE,2) **OVER**(**ORDER BY** ORD_DATE) **AS** NEXT_ORD_DAT
    FROM ORDERS;

ID	CUSTOMER_NAME	CITY	ORD_AMT	ORD_DATE	PREV_ORD_DAT	NEXT_ORD_DAT
1001	David Smith	GuildFord	10000.00	2017-04-01	NULL	2017-04-03
1002	David Jones	Arlington	20000.00	2017-04-02	NULL	2017-04-04
1003	John Smith	Shalford	5000.00	2017-04-03	2017-04-01	2017-04-05
1004	Michael Smith	GuildFord	15000.00	2017-04-04	2017-04-02	2017-04-06
1005	David Williams	Shalford	7000.00	2017-04-05	2017-04-03	2017-04-10
1006	Paum Smith	GuildFord	25000.00	2017-04-06	2017-04-04	2017-04-11
1007	Andrew Smith	Arlington	15000.00	2017-04-10	2017-04-05	2017-04-20
1008	David Brown	Arlington	2000.00	2017-04-11	2017-04-06	2017-04-25
1009	Robert Smith	Shalford	1000.00	2017-04-20	2017-04-10	NULL
1010	Peter Smith	GuildFord	500.00	2017-04-25	2017-04-11	NULL

Frame Specification

- A frame is a subset of the current partition, and the frame clause specifies how to define the subset
  - Frames are determined with respect to the current row
    - By defining a frame to be all rows from the partition start to the current row, one can compute running totals for each row
    - By defining a frame as extending *N* rows on either side of the current row, one can compute rolling averages
    - **ROWS**: The frame is defined by beginning and ending row positions (physical window)
    - **RANGE**: The frame is defined by rows within a value range (logical window)
    - **BETWEEN … AND …**: Specify both frame endpoints
      - **UNBOUNDED PRECEDING**: The bound is the first partition row
      - **UNBOUNDED FOLLOWING**: The bound is the last partition row
    - **CURRENT ROW**: For **ROWS**, the bound is the current row; For **RANGE**, the bound is the peers of the current row

Frame Specification Examples

- Sum over each partition
  - **SELECT** ID, CITY, ORD_AMT, ORD_DATE,
    AVG(ORD_AMT) **OVER**(**PARTITION BY** CITY **ORDER BY** ORD_DATE
    ROWS BETWEEN UNBOUNDED PRECEDING
    AND UNBOUNDED FOLLOWING
) **AS** AVG_AMT
    FROM ORDERS;

ID	CITY	ORD_AMT	ORD_DATE	AVG_AMT
1002	Arlington	20000.00	2017-04-02	12333.333333
1007	Arlington	15000.00	2017-04-10	12333.333333
1008	Arlington	2000.00	2017-04-11	12333.333333
1001	GuildFord	10000.00	2017-04-01	12625.000000
1004	GuildFord	15000.00	2017-04-04	12625.000000
1006	GuildFord	25000.00	2017-04-06	12625.000000
1010	GuildFord	500.00	2017-04-25	12625.000000
1003	Shalford	5000.00	2017-04-03	4333.333333
1005	Shalford	7000.00	2017-04-05	4333.333333
1009	Shalford	1000.00	2017-04-20	4333.333333

Frame Specification Examples

- A 2-record moving average
  - **SELECT** ID, CITY, ORD_AMT, ORD_DATE,
    AVG(ORD_AMT) **OVER**(**PARTITION BY** CITY **ORDER BY** ORD_DATE
    ROWS BETWEEN 1 **PRECEDING**
    AND 0 **FOLLOWING**
) **AS** AVG_AMT
    FROM ORDERS;

ID	CITY	ORD_AMT	ORD_DATE	AVG_AMT
1002	Arlington	20000.00	2017-04-02	20000.000000
1007	Arlington	15000.00	2017-04-10	17500.000000
1008	Arlington	2000.00	2017-04-11	8500.000000
1001	GuildFord	10000.00	2017-04-01	10000.000000
1004	GuildFord	15000.00	2017-04-04	12500.000000
1006	GuildFord	25000.00	2017-04-06	20000.000000
1010	GuildFord	500.00	2017-04-25	12750.000000
1003	Shalford	5000.00	2017-04-03	5000.000000
1005	Shalford	7000.00	2017-04-05	6000.000000
1009	Shalford	1000.00	2017-04-20	4000.000000

Frame Specification Examples

- A 2-record moving average
  - **SELECT** ID, CITY, ORD_AMT, ORD_DATE,
    AVG(ORD_AMT) **OVER**(**PARTITION BY** CITY **ORDER BY** ORD_DATE
    ROWS BETWEEN 1 **PRECEDING**
    AND CURRENT ROW
) **AS** AVG_AMT
    FROM ORDERS;

ID	CITY	ORD_AMT	ORD_DATE	AVG_AMT
1002	Arlington	20000.00	2017-04-02	20000.000000
1007	Arlington	15000.00	2017-04-10	17500.000000
1008	Arlington	2000.00	2017-04-11	8500.000000
1001	GuildFord	10000.00	2017-04-01	10000.000000
1004	GuildFord	15000.00	2017-04-04	12500.000000
1006	GuildFord	25000.00	2017-04-06	20000.000000
1010	GuildFord	500.00	2017-04-25	12750.000000
1003	Shalford	5000.00	2017-04-03	5000.000000
1005	Shalford	7000.00	2017-04-05	6000.000000
1009	Shalford	1000.00	2017-04-20	4000.000000

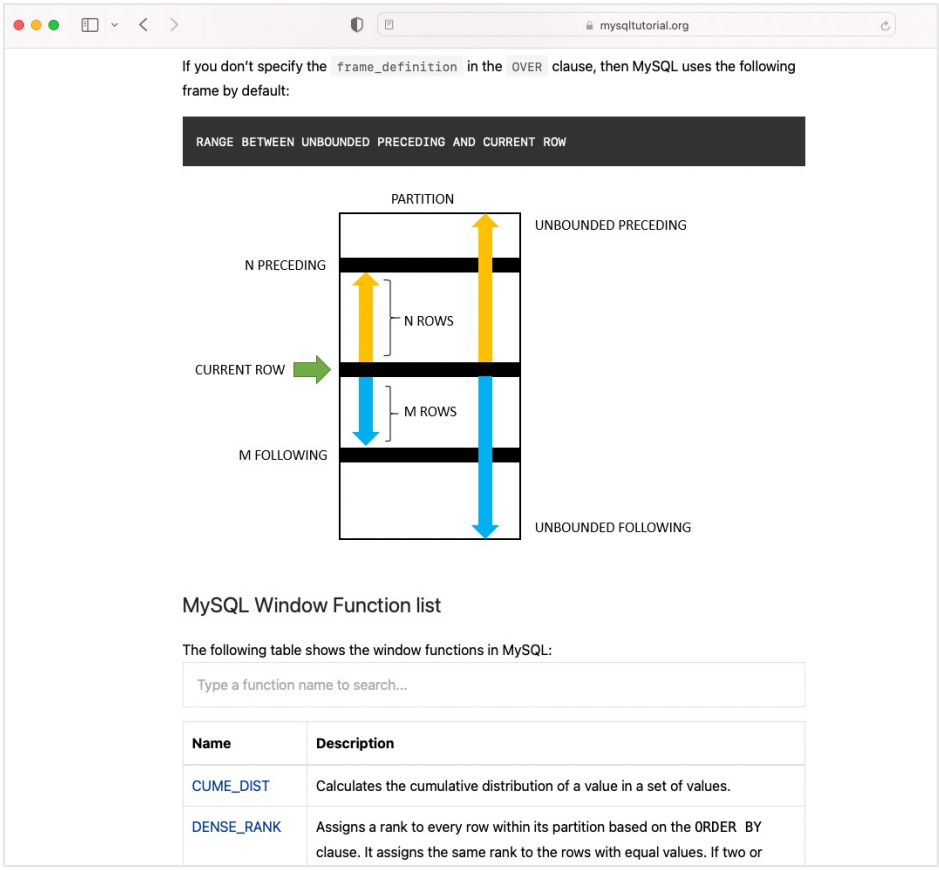
Frame Specification Examples

- A 3-day moving average
  - **SELECT** ID, ORD_DATE, ORD_AMT,
    AVG(ORD_AMT) **OVER**(**ORDER BY** ORD_DATE
    RANGE BETWEEN INTERVAL 2 **DAY PRECEDING**
    AND CURRENT ROW
) **AS** AVG_AMT
    FROM ORDERS;

ID	ORD_DATE	ORD_AMT	AVG_AMT
1001	2017-04-01	10000.00	10000.000000
1002	2017-04-02	20000.00	15000.000000
1003	2017-04-03	5000.00	11666.666667
1004	2017-04-04	15000.00	13333.333333
1005	2017-04-05	7000.00	9000.000000
1006	2017-04-06	25000.00	15666.666667
1007	2017-04-10	15000.00	15000.000000
1008	2017-04-11	2000.00	8500.000000
1009	2017-04-20	1000.00	1000.000000
1010	2017-04-25	500.00	500.000000

References

- A userful reference (free online tutorials): https://www.mysqltutorial.org/mysql-window-functions/



Agenda

- Window functions
- **Transactions**
  - Concept and examples
  - Levels of transactions

Transactions

- A transaction
  - is a "unit" of work
  - consists of *a sequence of query and/or update statements*

Transactions

- Why transactions?
  - Database systems are normally being accessed by many users or processes at the same time
    - Both queries and modifications
  - Unlike operating systems, which *support* interaction of processes, a DMBS needs to keep processes from troublesome interactions

Transactions

- *E.g.*, Bank

Transactions

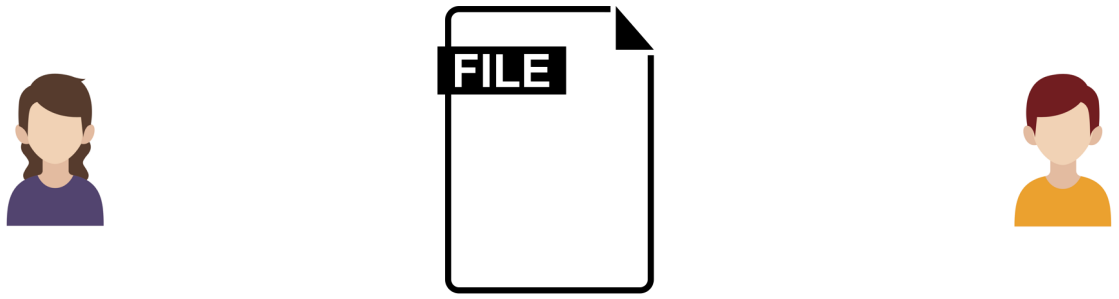
- *E.g.*, Bank

```
UPDATE accounts
SET balance = balance + 100
WHERE accNo = 456;
```

```
UPDATE accounts
SET balance = balance - 100
WHERE accNo = 123;
```

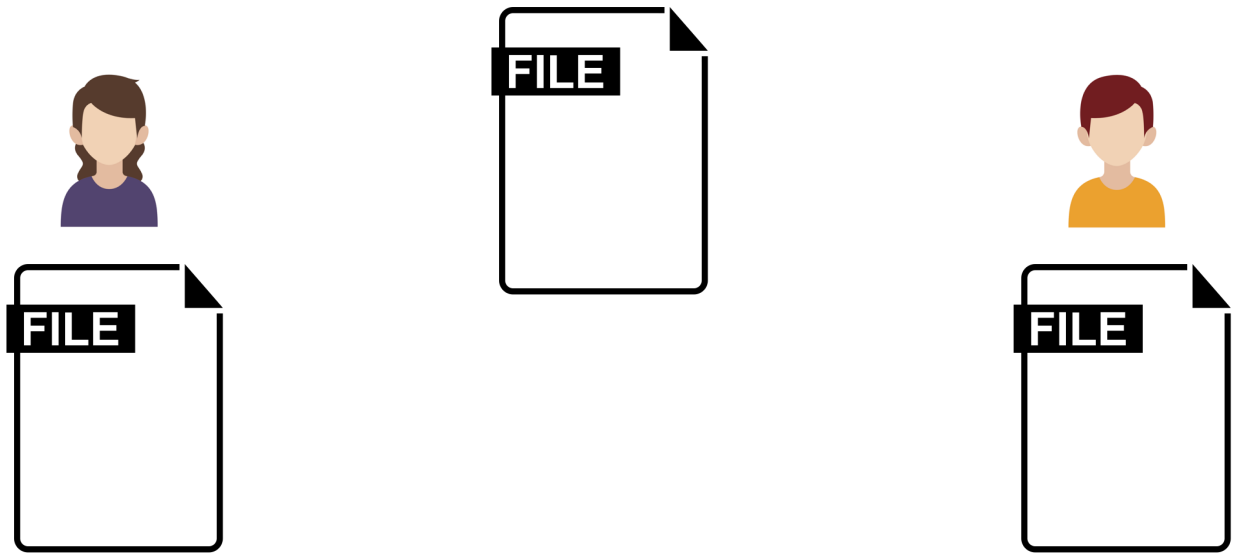
Transactions

- *C.f.*, File management in an OS



Transactions

- *C.f.*, File management in an OS



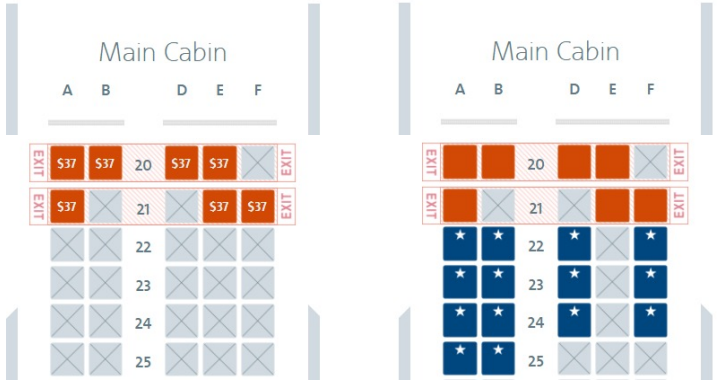
Transactions

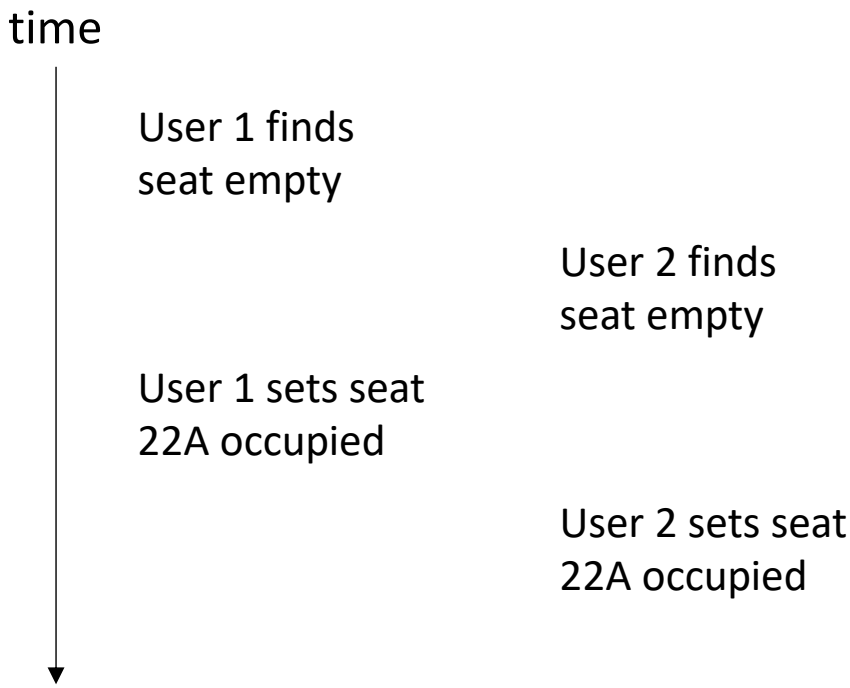
- Take home messages from the previous examples
  - You and your domestic partner each take $100 from different ATM's at about the same time
    - The DBMS would better make sure one account deduction does not get lost
  - *C.f.*, An OS allows two people to edit a document at the same time. If both write, one's changes get lost

Serializable Behaviors

- When there are more than one operations overlap in time, affecting the same data source
  - Each operation could perform correctly
  - While the global result might not be correct

- *E.g.*, Flight reservation





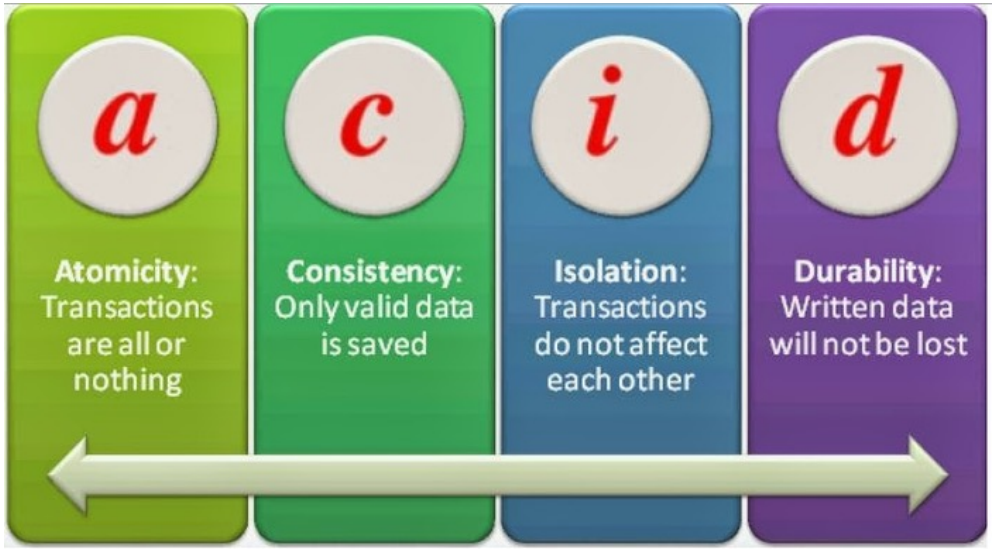
Serializable Behaviors

- When there are more than one operations overlap in time, affecting the same data source
  - Each operation could perform correctly
  - While the global result might not be correct

- SQL allows the programmer to state that certain operations must be serializable with respect to other operations
  - Operations must behave "as if" they were run *serially* – one at a time, with no overlap

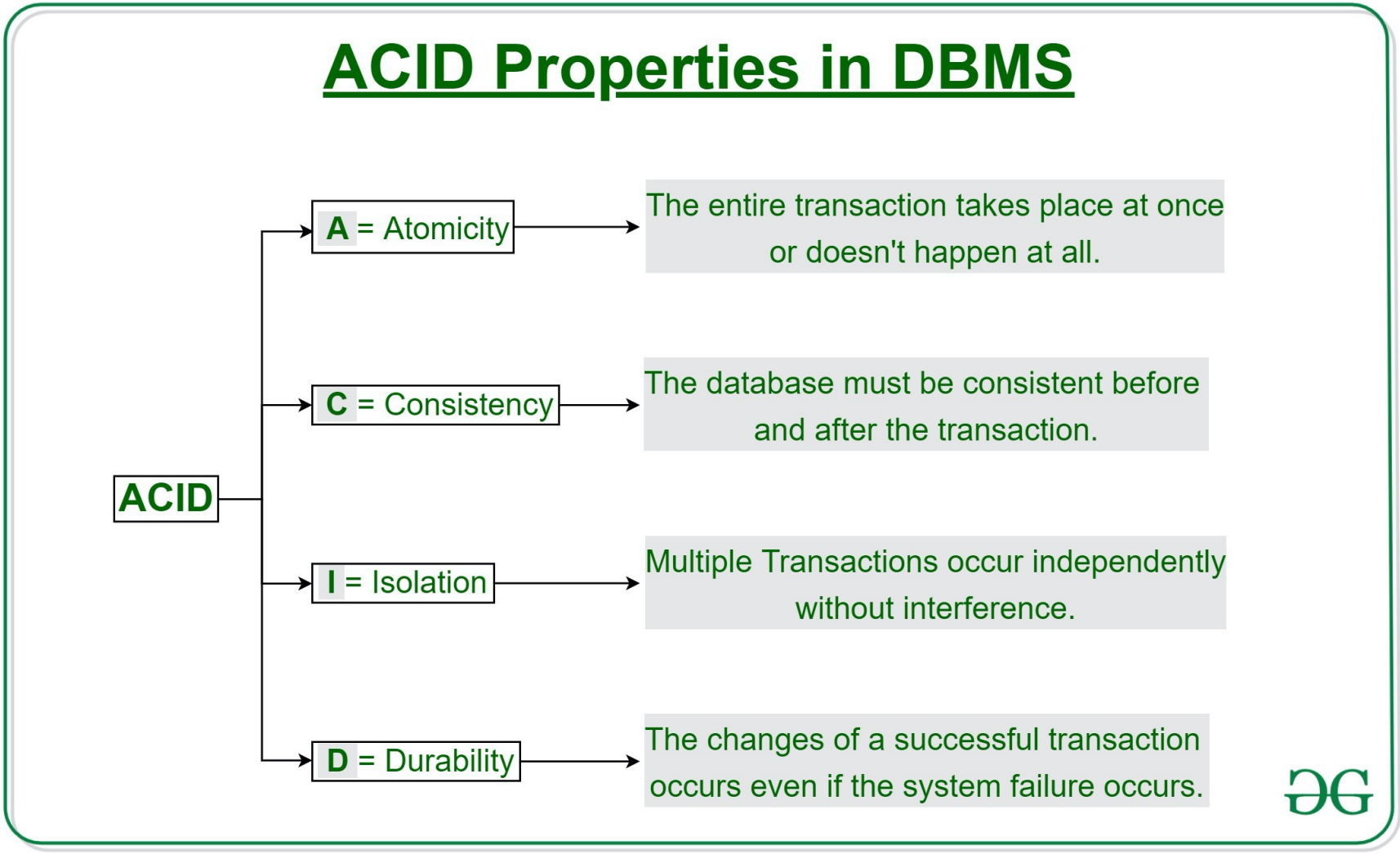
ACID Properties

- ACID properties
  - Atomic: Either fully executed or rolled back as if it never occurred
  - Consistent: Database constraints preserved
  - Isolated: Isolation from concurrent transactions – It appears to the user as if only one process executes at a time
  - Durable: Effects of a process survive a crash



* Image src: https://morpheusdata.com/blog/2015-01-29-when-do-you-need-acid-compliance

ACID Properties



* Image src: https://www.geeksforgeeks.org/acid-properties-in-dbms/

Transactions

- A transaction consists of *a sequence of query and/or update statements* and is a "unit" of work
  - To address both serialization and atomicity, group database operations into transactions
  - A transaction is a collection of one or more operations on DB that must be executed atomically
  - Transactions are executed in a serializable manner

Transactions

- *Transaction* = process involving database queries and/or modification
  - A transaction begins implicitly when an SQL statement is executed (the SQL standard)
  - The transaction must end with one of the following statements:
    - Commit work: The updates performed by the transaction become permanent in the database
    - Rollback work: All the updates performed by the SQL statements in the transaction are undone

Transactions

- Transactions could be formed by explicit programmer controls

  START TRANSACTION;

  …

  COMMIT; or **ROLLBACK**;

- **START TRANSACTION** statement is to declare that the guarded queries are of a group of operations that must be executed atomically
  - Each SQL statement that does not belong to any transaction explicitly is *a transaction with the single statement*

Transactions

- Transactions could be formed by explicit programmer controls

  START TRANSACTION;

  ...

  COMMIT; or **ROLLBACK**;

- **COMMIT** or **ROLLBACK** declares the end of a transaction
  - **COMMIT** causes a transaction to complete
    - The database modifications are now permanent in the database
  - **ROLLBACK** ends the transaction by aborting
    - No effects on the database
    - Failures like division by 0 or a constraint violation can also cause rollback, even if the programmer does not request it

Transactions

- *E.g.*, Bank

- **START TRANSACTION**;
  UPDATE accounts **SET** balance = balance + 100 **WHERE** accNo = 456;
  UPDATE accounts **SET** balance = balance - 100 **WHERE** accNo = 123;
  COMMIT;

A Transaction Example

- **START TRANSACTION**;
  SELECT @A:=SUM(salary) **FROM** *instructor* **WHERE** dept_name='Comp. Sci.';
  UPDATE budget_summary **SET** summary=@A **WHERE** dept_name='Comp. Sci.';
  COMMIT;

  - *C.f.*, Session variable - *@var_name*
    - Usages
      - **SET** *@var_name* = value or **SET** *@var_name* := value
      - *@var_name* := value in a SQL statement
    - Declaration is not required
    - Data type: Defined in the assignment
    - Scope: Until the end of the current session

Examples

```
-- Initialize to string
SET @id = 'A';

SELECT CONCAT(@id, 'B’);
-- Result: AB

-- Assign a number to the
-- same session variable
SET @id = 13;

SELECT @id * 3;
-- Result: 39

SELECT CONCAT(@id, 'B’);
-- Result: 13B
```

* Source: http://www.sqlines.com/mysql/session_variables

Another Transaction Example

- **SELECT** * **FROM** sales_history;

code	sales	month
A103	101	4
A102	54	5
A104	181	4
A101	184	4
A101	300	5
A103	17	5
A102	200	6
A104	87	6

Another Transaction Example

- **SELECT * FROM** sales_history;
- **START TRANSACTION**;

code	sales	month
A103	101	4
A102	54	5
A104	181	4
A101	184	4
A101	300	5
A103	17	5
A102	200	6
A104	87	6

Another Transaction Example

- **SELECT * FROM** sales_history;
- **START TRANSACTION**;
- **DELETE FROM** sales_history;
- **SELECT * FROM** sales_history;

code	sales	month

Another Transaction Example

- **SELECT** * **FROM** sales_history;
- **START TRANSACTION**;
- **DELETE FROM** sales_history;
- **SELECT** * **FROM** sales_history;
- **ROLLBACK**;
- **SELECT** * **FROM** sales_history;

code	sales	month
A103	101	4
A102	54	5
A104	181	4
A101	184	4
A101	300	5
A103	17	5
A102	200	6
A104	87	6

Yet Another Transaction Example

- Example: Interacting process
  - Assume a usual Sells(store, chocobar, price) relation, and suppose that Joe's Store sells only Snickers for $1.00 and Twix for $1.50
  - Sally is querying Sells for the highest and lowest price Joe charges
  - Joe decides to stop selling Snickers and Twix, but to sell only M&M's at $2.00

Yet Another Transaction Example

- Example: Interacting process
  - Sally's Program
    - Sally executes the following two SQL statements called (min) and (max) to help us remember what they do
      - (max)
        ```
        SELECT MAX(price) FROM Sells
        WHERE store = 'Joe''s Store';
        ```
      - (min)
        ```
        SELECT MIN(price) FROM Sells
        WHERE store = 'Joe''s Store';
        ```
  - Joe's Program
    - At about the same time, Joe executes the following steps: (del) and (ins)
      - (del)
        ```
        DELETE FROM Sells
        WHERE store = 'Joe''s Store';
        ```
      - (ins)
        ```
        INSERT INTO Sells
        VALUES('Joe''s Store', 'M&M's', 2.00);
        ```

Yet Another Transaction Example

- Example: Interacting process
  - Interleaving of Statements
    - Although (max) must come before (min), and (del) must come before (ins), there are no other constraints on the order of these statements
      - Unless we group Sally's and/or Joe's statements into transactions
  - Strange interleaving
    - Suppose the steps execute in the order (max)(del)(ins)(min)
      - Joe's Prices: {1.00, 1.50} {1.00, 1.50} {2.00}
      - Statement: (max) (del) (ins) (min)
      - Result: 1.50 2.00
    - Sally sees MAX < MIN

Yet Another Transaction Example

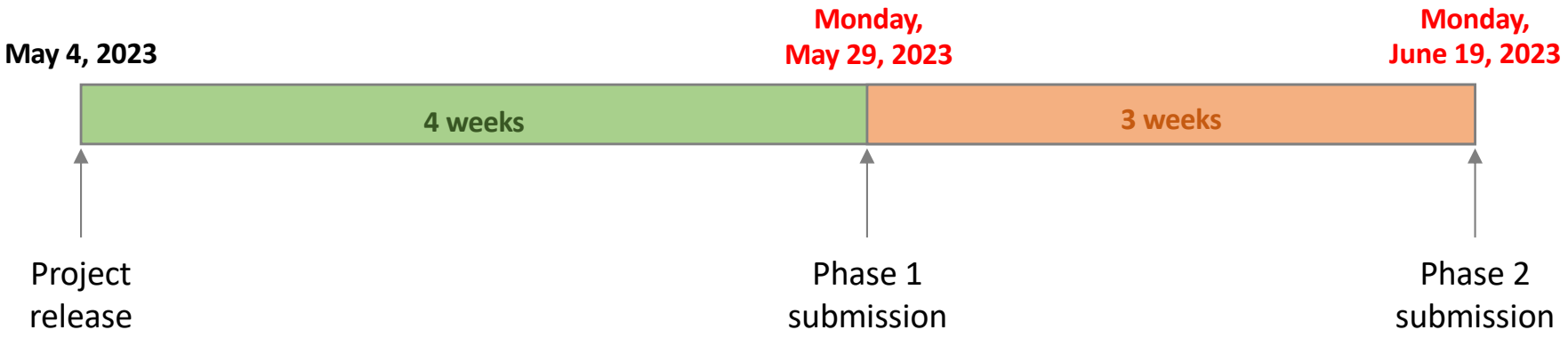
- Example: Interacting process
  - Fixing the Problem by Using Transactions
    - If we group Sally's statements (max)(min) into one transaction, then she cannot see the previous inconsistency
    - She sees Joe's prices at some fixed time
      - Either before or after he changes prices, or in the middle, but the MAX and MIN are computed from the same prices

Yet Another Transaction Example

- Example: Interacting process
  - Another Problem: Rollback
    - Suppose Joe executes (del)(ins), not as a transaction, but after executing these statements, thinks better of it and issues a ROLLBACK statement
    - If Sally executes her statements after (ins) but before the rollback, she sees a value, 2.00, that never existed in the database
  - Solution
    - If Joe executes (del)(ins) as a transaction, its effect cannot be seen by others until the transaction executes COMMIT
    - If the transaction executes ROLLBACK instead, then its effects can *never* be seen

Early-Bird Submission By Tonight

- Early submissions will earn extra credits



  - Phase 1 “space” submission: Monday, May 29, 2023
    - + 5% extra credit due: Monday, May 22, 2023
  - Phase 2 “time” submission: Monday, June 19, 2023

Phase 2: Database Optimization

- Goal: Design and implement a database instance that is efficient in time
  - You may also want to go through denormalization processes on your database instance from Phase 1
  - You may need to add indexes to the database

Phase 2: Database Optimization

- Denormalization
  - Usually carried out to improve the read performance of the database
    - Write may become slower
  - “Normalize until it hurts, denormalize until it works”

Revisited: Denormalization for Performance

- We may want to use non-normalized schema for performance

- Example: displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
  - Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
    - faster lookup
    - extra space and extra execution time for updates
    - extra coding work for programmer and possibility of error in extra code
  - Alternative 2: Use a materialized view defined a *course* ⋈ *prereq*
    - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Phase 2: Database Optimization

	ICUSTAY_ID	DRUG	DOSE_VAL_RX	DOSE_UNIT_RX	ROUTE
1	[redacted]	Amoxicillin-Clavulanic Acid	250	mg	PO
2	[redacted]	Amoxicillin	1000	mg	PO
3	<null>	Amoxicillin-Clavulanic Acid	500	mg	PO
4	<null>	CefazoLIN	2	g	IV
5	[redacted]	CefazoLIN	2	g	IV
6	[redacted]	Cefazolin	2	gm	IV
7	<null>	CefazoLIN	2	g	IV

- A query: **SELECT** ICUSTAY_ID, DRUG, DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
  FROM PRESCRIPTIONS P
  WHERE P.DRUG **LIKE** 'amoxicillin%' **OR** P.DRUG **LIKE** 'cefazolin';

- Without indexes on DRUG

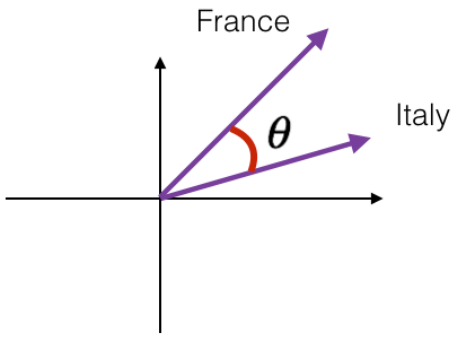
```
[2021-05-22 22:43:43] 500 rows retrieved starting from 1 in 3 s 935 ms
(execution: 3 s 841 ms, fetching: 94 ms)
```

- With an index on DRUG

```
[2021-05-22 22:53:41] 500 rows retrieved starting from 1 in 410 ms
(execution: 371 ms, fetching: 39 ms)
```

Phase 2: Database Optimization

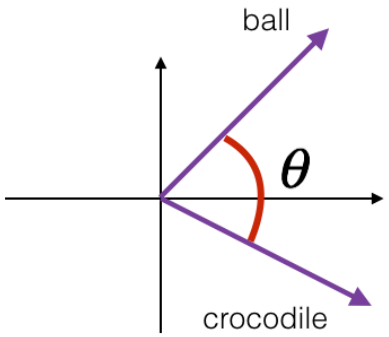
- Background: similarity
  - Contains the similarity (cosine similarity) between each pair of data instances



France and Italy are quite similar

θ is close to 0°

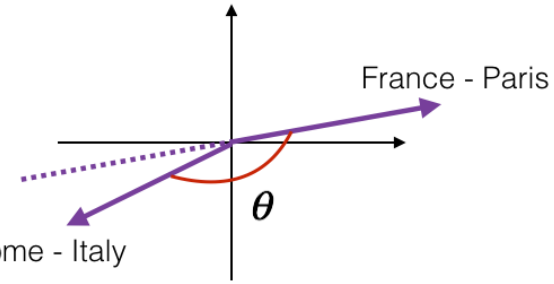
$\cos(\theta) \approx 1$



ball and crocodile are not similar

θ is close to 90°

$\cos(\theta) \approx 0$



the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)

θ is close to 180°

$\cos(\theta) \approx -1$

* Image src: https://datascience-enthusiast.com/DL/Operations_on_word_vectors.html

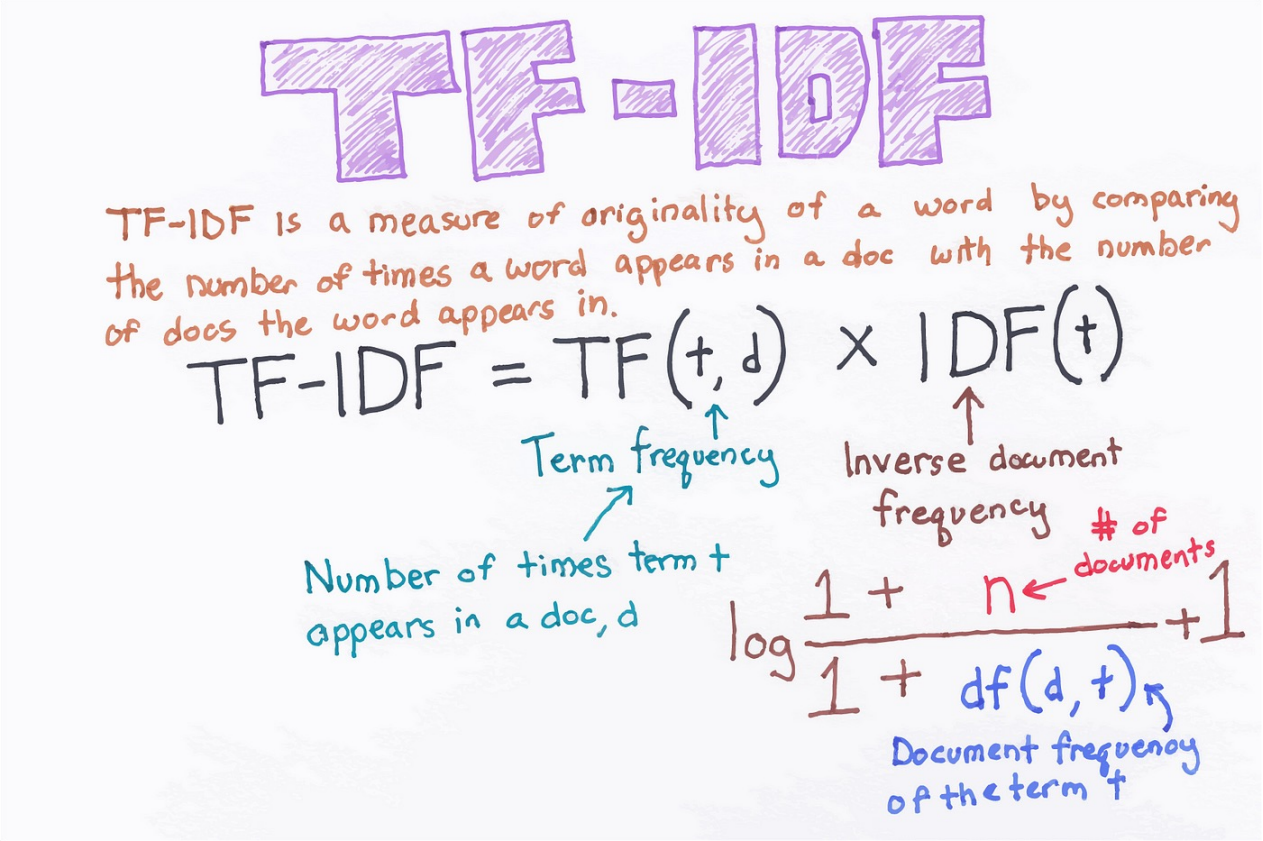
Phase 2: Database Optimization

- Background: similarity
  - Contains the similarity (cosine similarity) between each pair of data instances
  - We use it to measure the similarity between each pair of documents

docID	rcmdDocID	Score
10030990067319472539	10030990067319472539	1.0000000000000002
10030990067319472539	10043047191793211293	0.1329758471186704
10030990067319472539	10046263945557091965	0.03193380184794579
10030990067319472539	10046442708758033928	0.0987590617876676
10030990067319472539	10055720692083007959	0.1822967727883514
10030990067319472539	10056260562668352212	0.14230154872067327
10030990067319472539	10082680622925682	0.10731904324528363
10030990067319472539	10083204039233583851	0.18898456260537214
10030990067319472539	10090285731741476390	0
10030990067319472539	10110208171198839861	0.05863854143310857
10030990067319472539	10146231429070036509	0.06463178421565242
10030990067319472539	10193093611430635245	0.07202188938888371
10030990067319472539	10196528194641373645	0.010952017003927252
10030990067319472539	10238602817808319169	0.04562877191564491
10030990067319472539	10284149192682678031	0.03247219967020281

Phase 2: Database Optimization

- Background: frequency



* Image src: https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

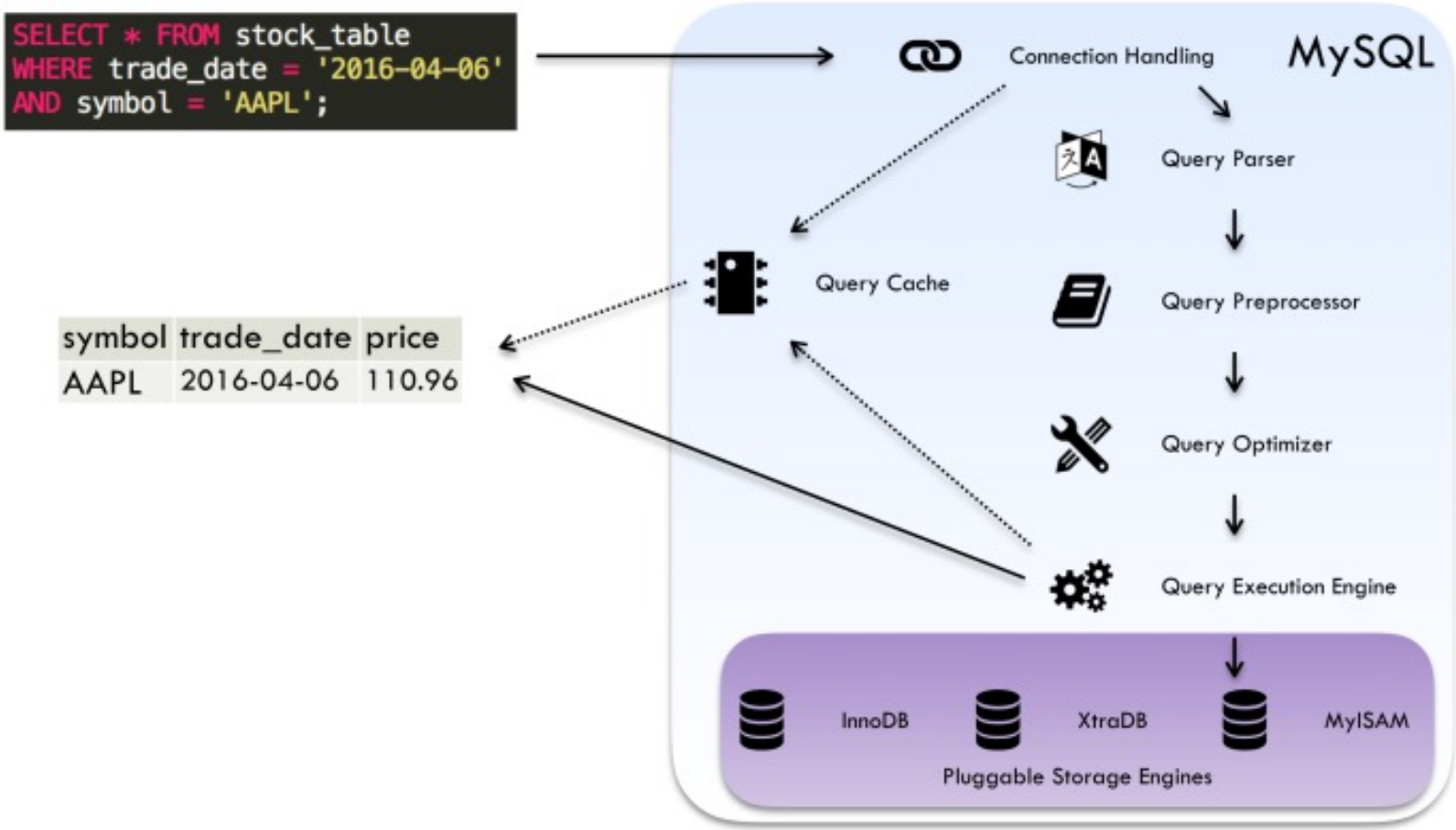
Phase 2: Database Optimization

- Background: frequency
  - Contains the TF-IDF (term frequency-inverse document frequency) scores for selected words for each document
  - Measures how relevant a word is to a document in a collection of documents
    - The higher the score, the more relevant that word is in that document
    - Example: https://sci2lab.github.io/ml_tutorial/tfidf/

docID	docTitle	tfidfWord	Score
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가공	0.004478425513887752
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가능	0.042000969424066226
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가속	0.005095251415320924
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가액	0.009833360305863292
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가임	0.009276335805335717
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가정	0.020639077897537134
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가족	0.057368035701677596
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가족법	0.008750184322130957
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	가치관	0.005245828309475467
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	각종	0.007061676068532896
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	간주	0.004251216381898961
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	갈등	0.02004565833769511
10011697067070999700	(2011) 북한이탈주민의 사회적응에 영향을 미치는 요인 분석 : 인천지역을 중심…	감금	0.009025864008681217

Phase 2: Database Optimization

- Handling SQL queries



Phase 2: Database Optimization

- Analysis on queries
  - EXPLAIN
  - SHOW PROFILES / SHOW PROFILE;