ECE30030/ITP30010 Database Systems

# Advanced SQL

*Reading: Chapters 4-5*

***Charmgil Hong***

charmgil@handong.edu

Spring, 2025

Handong Global University

# Agenda

- Join
- **Views**
- Window functions
- Keys

# Views

- It is not always desirable for all users to see the entire logical model of data
  - *E.g.*, consider a user who needs to know an instructor name and department, but not the salary
    - ➔ This user only needs to see the following relation (in SQL):
      - **SELECT** *ID*, *name*, *dept_name*
        **FROM** *instructor*

- View: provides a mechanism to hide certain data from the view of certain users
  - A view is a relation defined in terms of stored tables (called *base tables*) and other views
  - Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a view

# Views

- Syntax:

  **CREATE VIEW** *v* **AS** < query expression >

  where <query expression> is any legal SQL expression, and *v* represents the view name

  - Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
  - View definition is not the same as creating a new relation
  - A view definition causes the saving of an expression; the expression is substituted into queries using the view

# View Examples

- A view of instructors without their salary:
  - **CREATE VIEW** *faculty* ***AS***
    **SELECT** *ID*, *name*, *dept_name*
    **FROM** *instructor*

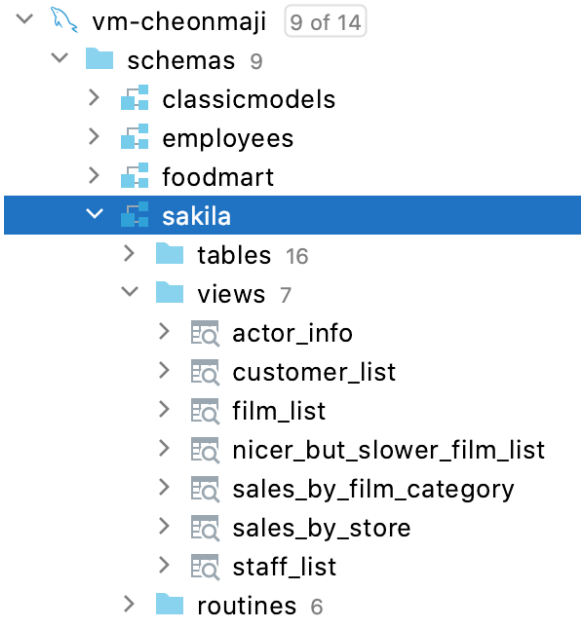
- Querying on a view is also possible:
  - **SELECT** *name*
    **FROM** *faculty*
    **WHERE** *dept_name* = 'Biology'

  - *C.f.*, find all instructors in the Biology department:
    - **SELECT** *name*
      **FROM** *instructor*
      **WHERE** *dept_name* = 'Biology'

# View Examples

- The attribute names of a view can be specified explicitly
  - **CREATE VIEW** *departments_total_salary(dept_name, total_salary)* **AS**
    **SELECT** *dept_name*, **SUM**(*salary*)
    **FROM** *instructor*
    **GROUP BY** *dept_name*;
    - Since the expression **SUM**(*salary*) does not have a name, the attribute name is specified explicitly in the view definition

- The *sakila* database, in the Class VM image, includes 7 sample views



# View Expansion

- View expansion: A way to define the meaning of views defined in terms of other views
  - Let view $v_1$ be defined by an expression $e_1$ that may itself contain uses of view relations
  - View expansion of an expression repeats the following replacement step:

    **repeat**

    Find any view relation $v_i$ in $e_1$

    Replace the view relation $v_i$ by the expression defining $v_i$

    **until** no more view relations are present in $e_1$
  - As long as the view definitions are not recursive, this loop will terminate

# Views Defined Using Other Views

- One view may be used in the expression defining another view
  - A view relation $v_1$ is said to *depend directly* on a view relation $v_2$ if $v_2$ is used in the expression defining $v_1$
  - A view relation $v_1$ is said to *depend on* view relation $v_2$ if either $v_1$ depends directly to $v_2$ or there is a path of dependencies from $v_1$ to $v_2$
  - A view relation $v$ is said to be *recursive* if it depends on itself

# Views Defined Using Other Views

- Examples
  - **CREATE VIEW** *physics_fall_2017* **AS**
    **SELECT** *course.course_id, sec_id, building, room_number*
    **FROM** *course, section*
    **WHERE** *course.course_id = section.course_id*
    **AND** *course.dept_name* = 'Physics'
    **AND** *section.semester* = 'Fall'
    **AND** *section.year* = '2017';

  - **CREATE VIEW** *physics_fall_2017_watson* **AS**
    **SELECT** *course_id, room_number*
    **FROM** *physics_fall_2017*
    **WHERE** *building*= 'Watson';

# Views Defined Using Other Views

- Both queries are equivalent (view expansion):
  - **CREATE VIEW** *physics_fall_2017_watson* **AS**
    **SELECT** *course_id*, *room_number*
    **FROM** *physics_fall_2017*
    **WHERE** *building*= 'Watson';

  - **CREATE VIEW** *physics_fall_2017_watson* **AS**
    **SELECT** *course_id*, *room_number*
    **FROM** (**SELECT** *course.course_id, sec_id, building, room_number*
    **FROM** *course, section*
    **WHERE** *course.course_id = section.course_id*
    **AND** *course.dept_name* = 'Physics'
    **AND** *section.semester* = 'Fall'
    **AND** *section.year* = '2017')
    **WHERE** *building*= 'Watson';

# Materialized Views

- Two kinds of views
  - Virtual: not stored in the database; just a query for constructing the relation
  - Materialized: physically constructed and stored

- Materialized view: pre-calculated (materialized) result of a query
  - Unlike a simple VIEW the result of a Materialized View is stored somewhere, generally in a table
  - Used when:
    - Immediate response is needed
    - The query where the Materialized View bases on would take to long to produce a result
  - Materialized Views must be refreshed occasionally

- MySQL does NOT support materialized views

# Update via a View

- Add a new tuple to *faculty* view which we defined earlier
  **INSERT INTO** *faculty* **VALUES** ('30765', 'Green', 'Music');
  - This insertion must be represented by the insertion into the *instructor* relation
    - Must have a value for salary

  - Must have a value for salary
    1) Reject the insert, OR
    2) Inset the tuple ('30765', 'Green', 'Music', null) into the *instructor* relation

# Update via a View

- Some updates cannot be translated uniquely
  - *E.g.*, **CREATE VIEW** *instructor_info* **AS**
    **SELECT** *ID*, *name*, *building*
    **FROM** *instructor*, *department*
    **WHERE** *instructor.dept_name* = *department.dept_name*;

    then, **INSERT INTO** *instructor_info*
    **VALUES** ('69987', 'White', 'Taylor');

  - Issues
    - Which department, if multiple departments are in Taylor?
    - What if no department is in Taylor?

    - On MySQL, an "*SQL error (1394): Can not insert into join view without fields list*" occurs

# Update via a View

- Example
  - **CREATE VIEW** *history_instructors* **AS**
    **SELECT** *
    **FROM** *instructor*
    **WHERE** *dept_name*='History';
- What happens if one inserts ('25566', 'Brown', 'Biology', 100000) into *history_instructors*?
  - **INSERT INTO** history_instructors
    **VALUES** ('25566', 'Brown', 'Biology', 100000)

| ID | name | dept_name | salary |
|---|---|---|---|
| 32343 | El Said | History | 60000.00 |
| 58583 | Califieri | History | 62000.00 |

| ID | name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 12121 | Wu | Finance | 90000.00 |
| 15151 | Mozart | Music | 40000.00 |
| 22222 | Einstein | Physics | 95000.00 |
| 25566 | Brown | Biology | 100000.00 |
| 32343 | El Said | History | 60000.00 |
| 33456 | Gold | Physics | 87000.00 |
| 45565 | Katz | Comp. Sci. | 75000.00 |
| 58583 | Califieri | History | 62000.00 |
| 76543 | Singh | Finance | 80000.00 |
| 76766 | Crick | Biology | 72000.00 |
| 83821 | Brandt | Comp. Sci. | 92000.00 |
| 98345 | Kim | Elec. Eng. | 80000.00 |

# Update via a View

- Most SQL implementations allow updates only on simple views
  - The **FROM** clause has only one database relation
  - The **SELECT** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **DISTINCT** specification
  - Any attribute not listed in the **SELECT** clause can be set to null
  - The query does not have a **GROUP BY** or **HAVING** clause

# Agenda

- Join
- Views
- **Window functions**
- Keys

# Window Functions in SQL

- First introduced to standard SQL in 2003
- Built-in functions that define the relationships between records
  - *"A window function performs a calculation across a set of table rows that are somehow related to the current row…Behind the scenes, the window function is able to access more than just the current row of the query result" (PostgreSQL)*
  - One can find ranks, percentiles, sums/averages, row numbers, *etc*.
- For aggregation functions, one can implement moving sums, moving averages, *etc*.
  - One can change the window sizes using the **WINDOW_FUNCTION** clause
- Cannot be used together with a **GROUP BY** clause
  - Both **PARTITION** and **GROUP BY** partition the data and compute some statistics
  - Does not reduce the number of records in the result

# 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**()

# 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

# 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 |

# Running Examples

- You can DIY…

```
CREATE TABLE DEPT
       (DEPTNO INT,
        DNAME VARCHAR(14),
        LOC VARCHAR(13) );

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH',   'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES',      'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');

CREATE TABLE EMP (
 EMPNO               INT NOT NULL,
 ENAME               VARCHAR(10),
 JOB                 VARCHAR(9),
 MGR                 INT,
 HIREDATE            DATE,
 SAL                 DECIMAL(7,2),
 COMM                DECIMAL(7,2),
 DEPTNO              INT);

INSERT INTO EMP VALUES (7839,'KING','PRESIDENT',NULL,'81-11-17',5000,NULL,10);
INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,'81-05-01',2850,NULL,30);
INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,'81-05-09',2450,NULL,10);
INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,'81-04-01',2975,NULL,20);
INSERT INTO EMP VALUES (7654,'MARTIN','SALESMAN',7698,'81-09-10',1250,1400,30);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',7698,'81-02-11',1600,300,30);
INSERT INTO EMP VALUES (7844,'TURNER','SALESMAN',7698,'81-08-21',1500,0,30);
INSERT INTO EMP VALUES (7900,'JAMES','CLERK',7698,'81-12-11',950,NULL,30);
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',7698,'81-02-23',1250,500,30);
INSERT INTO EMP VALUES (7902,'FORD','ANALYST',7566,'81-12-11',3000,NULL,20);
INSERT INTO EMP VALUES (7369,'SMITH','CLERK',7902,'80-12-09',800,NULL,20);
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,'82-12-22',3000,NULL,20);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,'83-01-15',1100,NULL,20);
INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,'82-01-11',1300,NULL,10);
```

# Aggregation Example

- Sum over each manager
  - **SELECT** ENAME, SAL, MGR,
    **SUM**(SAL) **OVER** (**PARTITION BY** MGR) SUM_MGR
    **FROM** EMP;

| ENAME | SAL | MGR | SUM_MGR |
|---|---|---|---|
| KING | 5000.00 | NULL | 5000.00 |
| FORD | 3000.00 | 7566 | 6000.00 |
| SCOTT | 3000.00 | 7566 | 6000.00 |
| MARTIN | 1250.00 | 7698 | 6550.00 |
| ALLEN | 1600.00 | 7698 | 6550.00 |
| TURNER | 1500.00 | 7698 | 6550.00 |
| JAMES | 950.00 | 7698 | 6550.00 |
| WARD | 1250.00 | 7698 | 6550.00 |
| MILLER | 1300.00 | 7782 | 1300.00 |
| ADAMS | 1100.00 | 7788 | 1100.00 |
| BLAKE | 2850.00 | 7839 | 8275.00 |
| CLARK | 2450.00 | 7839 | 8275.00 |
| JONES | 2975.00 | 7839 | 8275.00 |
| SMITH | 800.00 | 7902 | 800.00 |

# Ranking Example

- The base query:
  - **SELECT** EMPNO, ENAME, SAL,
    **SUM**(SAL) **OVER**(**ORDER BY** SAL
    **ROWS BETWEEN UNBOUNDED PRECEDING**
    **AND UNBOUNDED FOLLOWING**) TOTSAL
    **FROM** EMP;

| EMPNO | ENAME | SAL | TOTSAL |
|---|---|---|---|
| 7369 | SMITH | 800.00 | 29025.00 |
| 7900 | JAMES | 950.00 | 29025.00 |
| 7876 | ADAMS | 1100.00 | 29025.00 |
| 7654 | MARTIN | 1250.00 | 29025.00 |
| 7521 | WARD | 1250.00 | 29025.00 |
| 7934 | MILLER | 1300.00 | 29025.00 |
| 7844 | TURNER | 1500.00 | 29025.00 |
| 7499 | ALLEN | 1600.00 | 29025.00 |
| 7782 | CLARK | 2450.00 | 29025.00 |
| 7698 | BLAKE | 2850.00 | 29025.00 |
| 7566 | JONES | 2975.00 | 29025.00 |
| 7902 | FORD | 3000.00 | 29025.00 |
| 7788 | SCOTT | 3000.00 | 29025.00 |
| 7839 | KING | 5000.00 | 29025.00 |

# Ranking Example

- A cumulative sum:
  - **SELECT** EMPNO, ENAME, SAL,
    **SUM**(SAL) **OVER**(**ORDER BY** SAL
    **ROWS BETWEEN UNBOUNDED PRECEDING**
    **AND CURRENT ROW**) TOTSAL
    **FROM** EMP;

| EMPNO | ENAME | SAL | TOTSAL |
|---|---|---|---|
| 7369 | SMITH | 800.00 | 800.00 |
| 7900 | JAMES | 950.00 | 1750.00 |
| 7876 | ADAMS | 1100.00 | 2850.00 |
| 7654 | MARTIN | 1250.00 | 4100.00 |
| 7521 | WARD | 1250.00 | 5350.00 |
| 7934 | MILLER | 1300.00 | 6650.00 |
| 7844 | TURNER | 1500.00 | 8150.00 |
| 7499 | ALLEN | 1600.00 | 9750.00 |
| 7782 | CLARK | 2450.00 | 12200.00 |
| 7698 | BLAKE | 2850.00 | 15050.00 |
| 7566 | JONES | 2975.00 | 18025.00 |
| 7902 | FORD | 3000.00 | 21025.00 |
| 7788 | SCOTT | 3000.00 | 24025.00 |
| 7839 | KING | 5000.00 | 29025.00 |

# Ranking Example

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

| 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** ENAME, SAL,
    **RANK() OVER** (**ORDER BY** SAL **DESC**) ALL_RANK,
    **DENSE_RANK() OVER** (**PARTITION BY** JOB **ORDER BY** SAL **DESC**) JOB_RANK
    **FROM** EMP;

| 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 |

# 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 |

# Aggregation Examples

- Sum over each manager
  - **SELECT** ENAME, SAL, MGR,
    **SUM**(SAL) **OVER** (**PARTITION BY** MGR) **AS** SUM_MGR
    **FROM** EMP;

| ENAME | SAL | MGR | SUM_MGR |
|---|---|---|---|
| KING | 5000.00 | NULL | 5000.00 |
| FORD | 3000.00 | 7566 | 6000.00 |
| SCOTT | 3000.00 | 7566 | 6000.00 |
| MARTIN | 1250.00 | 7698 | 6550.00 |
| ALLEN | 1600.00 | 7698 | 6550.00 |
| TURNER | 1500.00 | 7698 | 6550.00 |
| JAMES | 950.00 | 7698 | 6550.00 |
| WARD | 1250.00 | 7698 | 6550.00 |
| MILLER | 1300.00 | 7782 | 1300.00 |
| ADAMS | 1100.00 | 7788 | 1100.00 |
| BLAKE | 2850.00 | 7839 | 8275.00 |
| CLARK | 2450.00 | 7839 | 8275.00 |
| JONES | 2975.00 | 7839 | 8275.00 |
| SMITH | 800.00 | 7902 | 800.00 |

# Nonaggregation 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 |

# Nonaggregation 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 |

# Nonaggregation 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 |

# Nonaggregation 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 |

# Nonaggregation 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 |

# Nonaggregation 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 |

# Frame Specification Examples

- Valid *units* for **INTERVAL**

| ***unit* Value** | **Expected *expr* Format** |
|---|---|
| **MICROSECOND** | MICROSECONDS |
| **SECOND** | SECONDS |
| **MINUTE** | MINUTES |
| **HOUR** | HOURS |
| **DAY** | DAYS |
| **WEEK** | WEEKS |
| **MONTH** | MONTHS |
| **QUARTER** | QUARTERS |
| **YEAR** | YEARS |
| **SECOND_MICROSECOND** | 'SECONDS.MICROSECONDS' |
| **MINUTE_MICROSECOND** | 'MINUTES:SECONDS.MICROSECONDS' |
| **MINUTE_SECOND** | 'MINUTES:SECONDS' |
| **HOUR_MICROSECOND** | 'HOURS:MINUTES:SECONDS.MICROSECONDS' |
| **HOUR_SECOND** | 'HOURS:MINUTES:SECONDS' |
| **HOUR_MINUTE** | 'HOURS:MINUTES' |
| **DAY_MICROSECOND** | 'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS' |
| **DAY_SECOND** | 'DAYS HOURS:MINUTES:SECONDS' |
| **DAY_MINUTE** | 'DAYS HOURS:MINUTES' |
| **DAY_HOUR** | 'DAYS HOURS' |
| **YEAR_MONTH** | 'YEARS-MONTHS' |

Examples:

10 **PRECEDING**

**INTERVAL** 5 **DAY PRECEDING**

5 **FOLLOWING**

**INTERVAL** '2:30' **MINUTE_SECOND FOLLOWING**

Reference: https://dev.mysql.com/doc/refman/8.0/en/expressions.html#temporal-intervals

# Agenda

- Join
- Views
- Window functions
- **Keys**

# Keys

- Key: An attribute or a set of attributes, which help(s) uniquely identify a tuple of data in a relation

| EmployeeID | Name | Branch | Email |
|---|---|---|---|
| 10201 | Cooper | DBMI | cooper@institute.edu |
| 10203 | Abraham | DBMI | laboriel@institute.edu |
| 10204 | Abraham | CS | abe@institute.edu |
| 10207 | Elly | EE | elly@institute.edu |

  - Q: Which of the attributes can be a key?

# Keys

- Key: An attribute or a set of attributes, which help(s) uniquely identify a tuple of data in a relation
  - Why we need keys?
    - To force identity of data and
    - To ensure integrity of data is maintained
    - To establish relationship between relations
  - Types of Keys
    - Super key
    - Candidate key
    - Primary key
    - Alternate key
    - Foreign key
    - Composite key
    - Compound key
    - Surrogate key

# Super Keys

- Any possible unique identifier
- Any attribute or any set of attributes that can be used to identify tuple of data in a relation; *i.e.*, any of
  - Attributes with unique values or
  - Combinations of the attributes
  - *E.g.*,

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

# Candidate Keys

- Minimal subset of super key
  - If any proper subset of a super key is also a super key, then that (super key) cannot be a candidate key
  - *E.g.*,

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

➔ *EmployeeID*

*FileCD*

*Email*

*EmployeeID + FileCD*

*EmployeeID + Email*

*FileCD + Email*

*EmployeeID + FileCD + Email*

# Primary Keys (PKs)

- The candidate key chosen to uniquely identify each row of data in a relation
  - No two rows can have the same PK value
  - PK value cannot be NULL (every row must have a primary key value)

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

➔ *EmployeeID*

*FileCD* ***Pick any one as PK***

*Email*

# Alternate Keys

- The candidate keys that are NOT chosen as PK in a relation

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

➔ *EmployeeID*

*FileCD*

*Email*

*If we choose EmployeeID as PK,*
*then FileCD and Email become alternate keys*

# Foreign Keys

- An attribute in a relation that is used to define its relationship with another relation
  - Using foreign key helps in maintaining data integrity for tables in relationship

***Employee***

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

***Branch***

| Branch | Address |
|---|---|
| DBMI | 5607 Baum Blvd |
| CS | 260 S Bouquet St |
| EE | 3700 O'Hara St |
| BIO | 4249 Fifth Ave |

# Composite & Compound Keys

- Composite key: Any key with more than one attribute
  - *E.g.*,

| EmployeeID | FileCD | Name | Branch | Email |
|---|---|---|---|---|
| 10201 | D-201-C | Cooper | DBMI | cooper@institute.edu |
| 10203 | D-203-A | Abraham | DBMI | laboriel@institute.edu |
| 10204 | C-204-A | Abraham | CS | abe@institute.edu |
| 10207 | E-207-E | Elly | EE | elly@institute.edu |

➔ *EmployeeID + FileCD,* *EmployeeID + Email,* *FileCD + Email*

*EmployeeID + FileCD + Email*

- Compound key: A composite key that has at least one attribute, which is a foreign key
  - *E.g.*, Let us assume that we have defined a composite key (FileCD, Branch), it is also a compound key (considering the Branch table)

# Surrogate Keys

- If a relation has no attribute that can be used as a key, then we create an artificial attribute for this purpose
  - It adds no meaning to the data, but serves the sole purpose of identifying tuples uniquely in a table
    - ◯◯◯◯_ID with auto increment

# EOF

- Coming next:
  - Transactions