ECE30030/ITP30010 Database Systems

Advanced SQL

Reading: Chapters 4-5

Charmgil Hong

charmgil@handong.edu

Spring, 2025

Handong Global University

Announcements

- Term project is on
  - Due date: Jun 2, 2025
  - Early due: May 28, 2025

- HW#5 is posted (official release: May 22)
  - Due: Jun 5, 2025

Phase 1: Database Modeling

- Objective: Design and implement a database instance that is efficient in space
  - You are expected to conduct a database design using ERD and apply the normalization theory
    - Before the submission, each team is expected to run several iterations of design, implement, data import, and internal evaluation
  - We will check the correctness and completeness of your data by examining the output of the **views** suggested in next slides
  - The database size on the physical storage will be estimated; the smallest 10% teams will earn bonus points (**maximum +7%**)
  - Due date: Jun 2, 2025
    - Submission window: May 26-Jun 2 (*the submission page will open May 26*)
    - Early submission: Submissions made by May 28 will earn bonus points (**maximum +4%**). *START EARLY! SPEND MORE TIME ON PHASE 2!*

Provided Data

- `allrecords`
  - Collection of core meta-data about the full-text documents that KUBIC contains
  - Also contains the bulletin boards, user information, saved documents of each user
  - 1,255,795 records, 37 columns
  - Completely unnormalized; a lot of data anomalies

- `Frequency` (will be provided no later than May 19)
  - TF-IDF analysis of the service documents

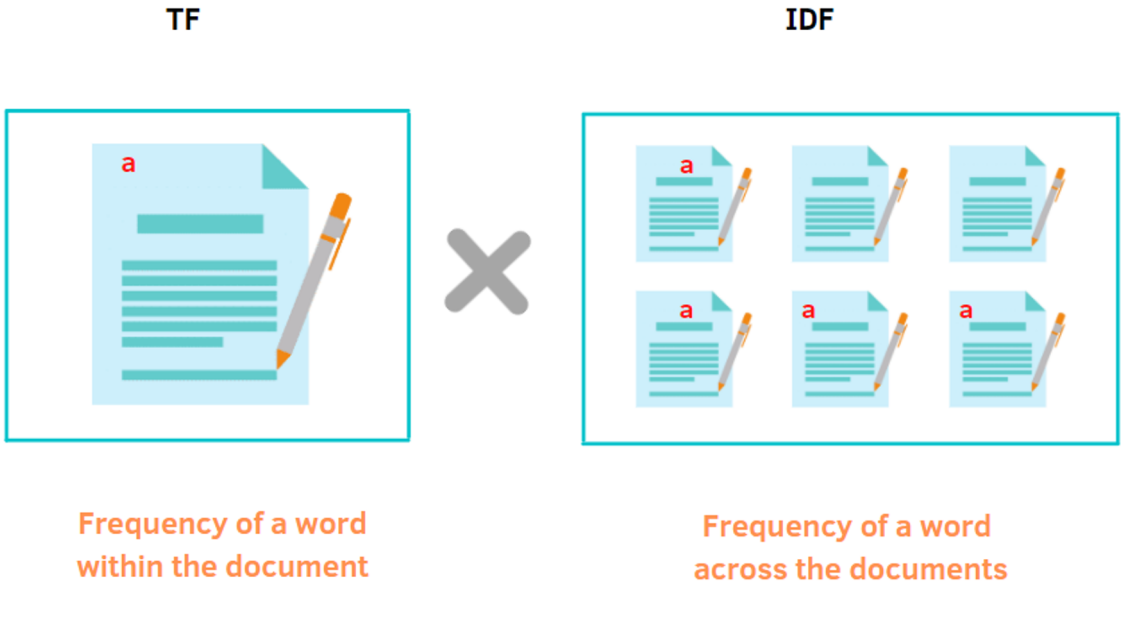
Provided Data

- frequency (tfidf scores)
  - TF-IDF analysis of the service documents
  - For extracting document keywords

	doc_id	doc_title	tfidf_word	score
1	10001499674539999232	[Testimony] Memories for 16 years	Winter	0.060791255336978
2	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Renovation	0.0499927602590186
3	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Construction	0.0541337674472854
4	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Conclusion	0.0605784847295904
5	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Reactor	0.1033610294564053
6	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Coexist	0.0763063389353876
7	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Relation	0.4369037285470069
8	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Improvement	0.0444494939093749
9	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Exchange	0.107860638059908
10	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	International	0.0445026309662148
11	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Keynote	0.0750398916694631
12	10001499674539999232	[Testimony] Memories for 16 years	Possibility	0.0077847399301015
13	10001499674539999232	[Testimony] Memories for 16 years	Malleable	0.0536955524993446
14	10001499674539999232	[Testimony] Memories for 16 years	Action	0.0309922260511834
15	10001499674539999232	[Testimony] Memories for 16 years	Opening	0.0147535028436665
16	10001499674539999232	[Testimony] Memories for 16 years	Development	0.0270588803000342
17	10001499674539999232	[Testimony] Memories for 16 years	Gaesung	0.0128351383024243
18	10001499674539999232	[Testimony] Memories for 16 years	Individual	0.0095602734398585
19	10001499674539999232	[Testimony] Memories for 16 years	Host	0.0717639131368882
20	10001499674539999232	[Testimony] Memories for 16 years	Verification	0.0138415646440895

Provided Data

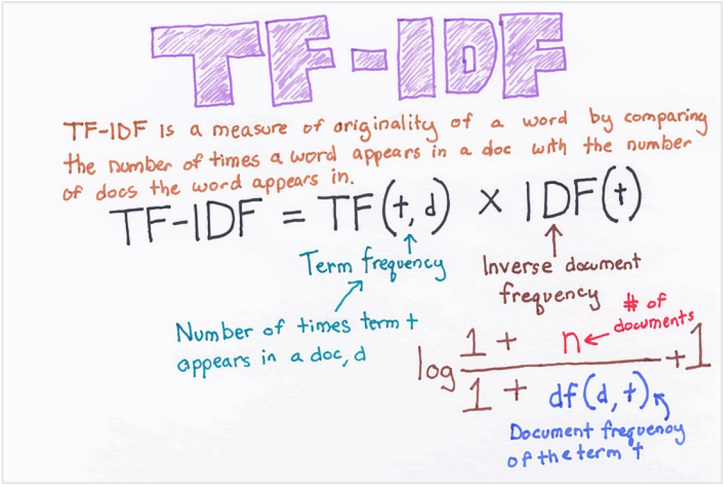
- `frequency` (tfidf scores)



Background

- TF-IDF stands for term frequency-inverse document frequency
  - Quantifies the importance or relevance of string (words, phrases, lemmas, *etc*.) in a document amongst a collection of documents (corpus)

- TF-IDF breaks down into two parts: TF and IDF
  - TF (term frequency): The weight of a term in a document that is simply proportional to the term frequency
  - IDF (inverse document frequency): The additional factor based on a corpus, to diminish the weight of terms that occur very frequently in the document set and to increase the weight of terms that occur rarely



Background

- 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
      - https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (EN)
      - https://wikidocs.net/31698 (KR)

	doc_id	doc_title	tfidf_word	score
1	10001499674539999232	[Testimony] Memories for 16 years	Winter	0.060791255336978
2	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Renovation	0.0499927602590186
3	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Construction	0.0541337674472854
4	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Conclusion	0.0605784847295904
5	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Reactor	0.1033610294564053
6	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Coexist	0.0763063389353876
7	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Relation	0.4369037285470069
8	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Improvement	0.0444494939093749
9	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Exchange	0.107860638059908
10	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	International	0.0445026309662148
11	1000120135460720000	Influence on Tenjin Regional Economic Growth of Foreign …	Keynote	0.0750398916694631

Phase 1: Database Modeling

- Views to create (and submit)
  - 8. View: doc_top3_topics
    - Show the 3 most common topics and the number of documents with each of the three topics, from the stored documents with post (post_date) year of 2021

    - Step 1: Filter documents by post year
    - Step 2: Aggregate TF-IDF scores across documents
      - The `frequency` table will be provided no later than May 19
    - Step 3: Count the number of documents for each of the top 3 topic words

Referential Integrity Constraints

- Foreign keys can be specified as part of the SQL DDL (CREATE TABLE) statement
  - *E.g.*, **FOREIGN KEY** (*dept_name*) **REFERENCES** *department*
- By default, a foreign key references the primary key attributes of the referenced table
- SQL allows a list of attributes to be referenced specified explicitly
  - *E.g.*, **FOREIGN KEY** (*dept_name*) **REFERENCES** *department* (*dept_name*)

Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation

- An alternative, in case of delete or update is to cascade
  - *E.g.*, **CREATE TABLE** *course* (
    (…
    dept_name **VARCHAR**(20),
    FOREIGN KEY (*dept_name*) **REFERENCES** *department*
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    . . .)

  - Instead of **CASCADE** we can use:
    - **SET NULL**
    - **SET DEFAULT**

Cascading Actions in Referential Integrity

- Definitions
  - **ON DELETE CASCADE**: If a key in the parent table is deleted, all corresponding rows in the child table are also deleted
  - **ON UPDATE CASCADE**: If a key in the parent table is updated, all corresponding foreign keys in the child table are also updated

  - Alternatives to **CASCADE**
    - **SET NULL**: Sets the foreign key column to NULL in the child when the parent row is deleted/updated
    - **SET DEFAULT**: Sets the foreign key column to its default value when the parent row is deleted/updated.

Cascading Actions in Referential Integrity

- Examples
  - Given **CREATE TABLE** department (
    dept_name **VARCHAR**(20) **PRIMARY KEY**);

    CREATE TABLE course (
    course_id **INT PRIMARY KEY**,
    course_name **VARCHAR**(50),
    dept_name **VARCHAR**(20),
    FOREIGN KEY (dept_name) **REFERENCES** department(dept_name)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
  - If one runs:
    - **DELETE FROM** department **WHERE** dept_name = 'CS';
      → All courses in the course table that reference dept_name = 'CS' are automatically deleted
    - **UPDATE** department **SET** dept_name = 'CSE' **WHERE** dept_name = 'CS';
      → All foreign key values in course.dept_name that were 'CS' are now 'CSE'

Cascading Actions in Referential Integrity

- Examples
  - Given **CREATE TABLE** department (
    dept_name **VARCHAR**(20) **PRIMARY KEY**);

    CREATE TABLE course (
    course_id **INT PRIMARY KEY**,
    course_name **VARCHAR**(50),
    dept_name **VARCHAR**(20) **DEFAULT** 'General',
    FOREIGN KEY (dept_name) **REFERENCES** department(dept_name)
    ON DELETE SET NULL
    ON UPDATE DEFAULT);
  - If one runs:
    - **DELETE FROM** department **WHERE** dept_name = 'CS';
      → All courses in the course table where dept_name = 'CS' will now have dept_name = NULL
    - **UPDATE** department **SET** dept_name = 'CSE' **WHERE** dept_name = 'CS';
      → All foreign key values in course.dept_name that were 'CS' are now 'General' (default)

Agenda

- Window functions

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

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

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

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

$$\text{CUME\_DIST}(r) = \frac{\text{Number of rows with value} \leq \text{value at row } r}{\text{Total number of rows in partition}}$$

$$\text{PERCENT\_RANK}(r) = \frac{\text{Rank}(r) - 1}{\text{Total rows in partition} - 1}$$

- 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