LEADS 3-Day Camp
Session 3: Data Management: Relational Databases and SQL

Il-Yeol Song, Ph.D.
Professor
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
Song@drexel.edu
http://www.cci.drexel.edu/faculty/song/

1

Hello!

2

Instructor

- Professor, College of Computing & Informatics, Drexel University,
- PhD in CS, LSU, Baton Rouge, USA, 1988
- CCI PhD Program Director (2010-2015)
- CVDI Deputy Director (2012-2014)
- Research Topics: *Conceptual Modeling, Data Warehousing, Big Data Management and Data Analytics, Smart Aging*
- Elected as an **ER Fellow,** 2012
- Named an **ACM Distinguished Scientist** in 2013
- Received **Peter Chen Award in Conceptual Modeling** in 2015
- Four teaching awards from Drexel (1991, 2000, 2001, 2011) including **Lindback Distinguished Teaching Award** (2001)
- **Co-Editor-in-Chief**, *Journal of Computing Science & Engineering*
- **Consulting Editor**, *Data & Knowledge Engineering*
- **Co-Chair**, **iSchool Model Data Science Curriculum Committee**
- **Chair, IEEE Big Data and Smart Computing (BigComp) Conference**
- Published about 200+ papers

Acknowledgement

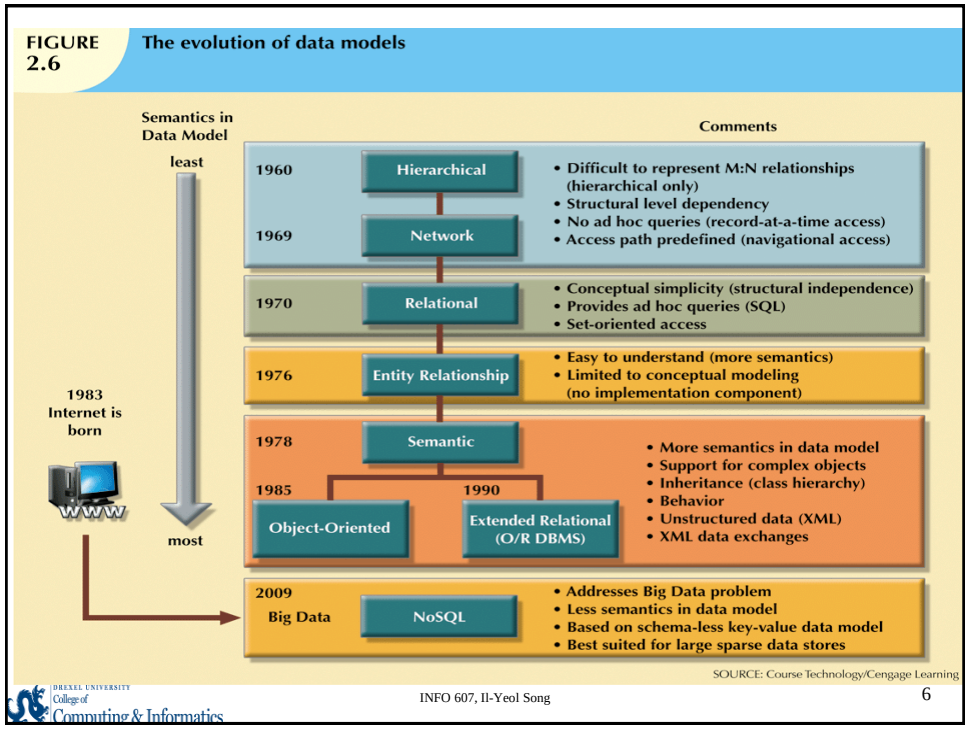
- Materials from:
  - Lecture Notes from Drexel CCI INFO 605 and 606 Class (Il-Yeol Song)
  - Some diagrams and examples from

    Carlos Coronel and Steven Morris, Database Systems: Design, Implementation, and Management, 13th Edition, 2017. Cengage Learning.

Contents

- Evolution of Data Models
- Relational Databases
- SQL
  - Basic SQL commands
  - SQL for Data Engineering
  - SQL for Analytics
- NoSQL Databases
- MongoDB Basic CRUD Operations

5



6

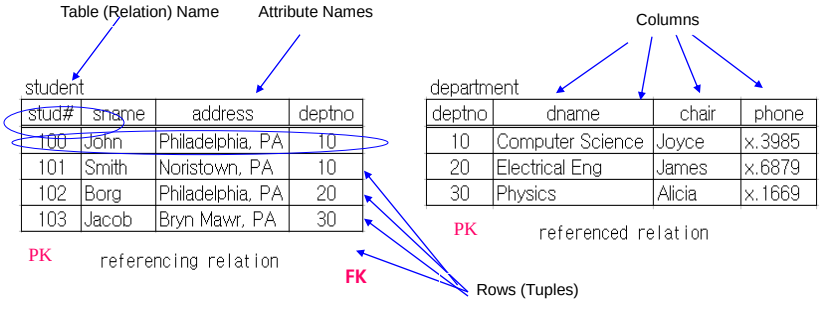
Basics in the Relational Model

- Data representation: a set of tables with structured data
- Primary keys (Candidate key, Alternate keys)
- Foreign Key
- Referential Integrity constrains
- Integrity constraints in RDBMSs (PK, RI, CHECK, Triggers)
- Null values
- Indexes
- Metadata and Data Dictionary
- Relational schema design using ER diagrams
- Normalization (FD, MVD, JD) and Normal Forms
- SQL (*Aggregation*, *Subquery*, *JOIN*, PL/SQL functions/procedures/triggers)
- ACID property in multi-user ***transactions***

Terminology in RDB

Relational Schema

STUDENT (**stud#,** sname, address, deptno)
DEPARTMENT (**deptno,** dname, chair, phone)



A **Primary Key (PK)** uniquely identifies each row.
A **Foreign Key (FK)** is a set of attributes
that are primary key values from another table.

Characteristics of Relational Model

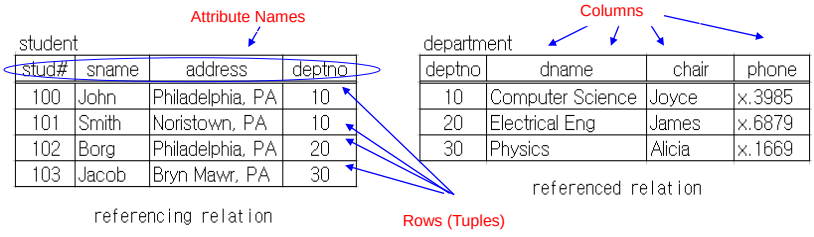


TABLE 3.1 Characteristics of a Relational Table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each row/column intersection represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or a combination of attributes that uniquely identifies each row.

9

Example of a DDL in SQL Defining a Relational Schema

student

stud#	sname	address	deptno
100	John	Philadelphia, PA	10
101	Smith	Noristown, PA	10
102	Borg	Philadelphia, PA	20
103	Jacob	Bryn Mawr, PA	30

referencing relation

department

deptno	dname	chair	phone
10	Computer Science	Joyce	x.3985
20	Electrical Eng	James	x.6879
30	Physics	Alicia	x.1669

referenced relation

```
CREATE TABLE Department (
        DeptNo          NUMBER(5)       PRIMARY KEY,
        DName           VARCHAR2(15)  NOT NULL      ,
        Chair           VARCHAR2 (20)               ,
        Phone           CHAR (10)                   ,
        );

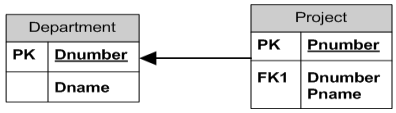
CREATE TABLE Student (
        Stud#           NUMBER(8)                   ,
        SName           VARCHAR2(20) NOT NULL       ,
        Address         VARCHAR2 (40)               ,
        DeptNo           NUMBER (5)                 ,
        CONSTRAINT Stud_PK  PRIMARY KEY (Stud#)     ,
        CONSTRAINT Stud_FK FOREIGN KEY (DeptNo)
                REFERENCES Department (DeptNo)
        );
```


10

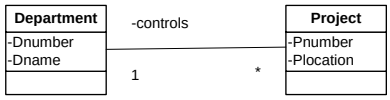
ER Model: Popular Notations

Chen

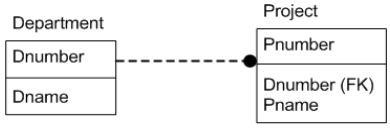
Visio 's Relational notation



UML

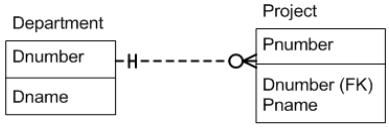


IDEF1X



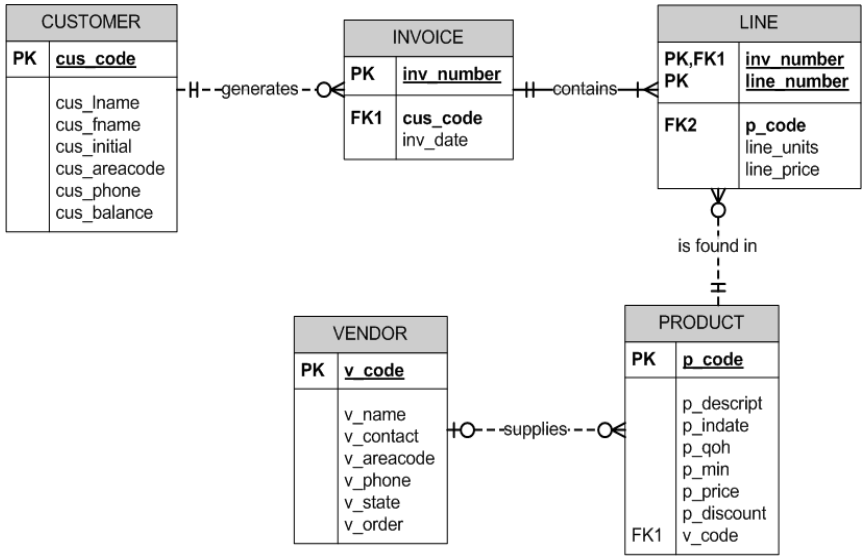
Crow's foot with Relational Notation in Visio

Visio IDEF1X with Crow's foot



11

ERD for Relational DBs



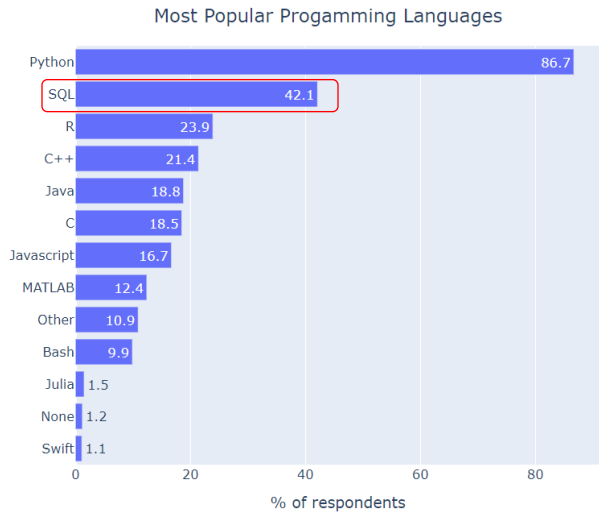
12

Why Relational Databases?

- Logical simplicity of the schema: **Tables**
- Easy, powerful, standard database language: **SQL**
- Transaction reliability: **ACID property** (Atomicity, Consistency, Isolation, Durability)

- Ad-hoc query processing
- Mature and reliable technologies
- Commercial investment for the last 40 years
- A large group of man-power and user bases

2020 Kaggle Data Science & Machine Learning Survey



https://www.kaggle.com/code/paultimothymooney/2020-kaggle-data-science-machine-learning-survey/notebook

Top Desired Skills in Data Science (2021)

11 Skills Data Scientists Need

1. Data Visualization
2. Python
3. SQL/NoSQL
4. Social Media Mining
5. Fundamental Statistics
6. Natural Language Processing/Machine Learning
7. Microsoft Excel
8. High-Level Math
9. Teamwork
10. Communication
11. Business Savvy

Source: https://bootcamp.berkeley.edu/blog/data-scientist-skills/

Introduction to SQL

- Pronounced 'see-quel'
- ANSI prescribes a standard SQL
- Developed by IBM in 1974, System R project
- First commercial implementation by Oracle (1976)
- ANSI/ISO Standard Versions:
  - SQL 86 (SQL 1)
  - SQL 89 (SQL 1 Level 2): RI integrity
  - SQL 92 (SQL 2)
  - SQL:1999 for OO features
  - SQL:2003 (XML features)
  - SQL: 2011 (Temporal data, windows functions)
  - SQL: 2016: JSON
  - SQL: 2019: Multi-dimensional Array (MDArray type and operations)
- Several dialects and extensions exist

SQL Topics

- **SQL Basics**
  - Basic DDL and DML
  - Simple commands on a Single table
  - Joins (inner join, outer join, natural joins, multi-table joins)
  - Subquery
- **SQL for Data Engineering**
  - Date manipulation and Date Arithmetic
  - String manipulation
  - Merge
- **SQL for Analytics**
  - Aggregation, grouping, and ordering
  - OLAP functions
  - Ranking, partition by, buckets
  - Creating new derived columns

Two major Components of SQL

Data Definition Language (DDL)

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE/DROP VIEW
- CREATE/DROP INDEX

Data Manipulation Language(DML)

- UPDATE TABLE
- INSERT INTO
- DELETE FROM
- **SELECT** FROM

DDL manages metadata
DML manages data

A Typical SQL Statement and Rules

STUDENT

ID	Name	EnrollYear

- SELECT enrollyear FROM student WHERE name ='John';
- SQL statement comprises
  - Keywords
    - **Case-insensitive**
    - Basic command set: fewer than 100 English words
    - Space-independent
  - User-defined word (table names, attribute names)
    - **Case-insensitive**
  - **Data**
    - **Case-sensitive** (In MS Access, case-insensitive)
  - Ends with a semi-colon

An Overview of SQL Commands

DDL

--Changing attribute type

ALTER TABLE project
MODIFY (budget NUMBER(9,2));

--Adding an attribute

ALTER TABLE project **ADD** (manager CHAR(10));

-- Delete a column from a table

ALTER TABLE project
DROP COLUMN manager;

--Removing a table from database

DROP TABLE project;

DML

--Inserting a row to a table

INSERT INTO project VALUES (1234, 'Perfect Project', NULL, 'John');

--Changing a value of attribute

UPDATE project SET budget = 1.1*budget WHERE projno > 1000;

--Deleting a row from a table

DELETE FROM project WHERE manager = 'John';

Queries using SELECT Statement

SELECT	**Specifies which columns are to appear in output**
FROM	**Specifies table(s) to be used**
WHERE	**Filters rows with conditions**
GROUP BY	**Forms groups of rows with the same column value.**
HAVING	**Filters groups subject to some condition.**
ORDER BY	**Specifies the order of the output.**

- ***Only SELECT and FROM are mandatory.***
- ***Order of the clauses cannot be changed.***

An Overview of Simple Commands

(1) **SELECT** * **FROM** Instructor;

(2) **SELECT DISTINCT** fName **FROM** Instructor;

(3) **SELECT** fName, lName, ssn **FROM** Instructor
WHERE deptCode = 'math';

(4) **SELECT** fName, lName **FROM** Instructor
WHERE bonus > 1000;

(5) **SELECT** * **FROM** Instructor
WHERE (bonus >= 500 **AND** bonus <= 1000);

(6) **SELECT** * **FROM** Instructor **WHERE** deptCode **IS NULL**;

(7) **SELECT** * **FROM** Instructor **WHERE** deptCode **IS NOT NULL**;

(8) **SELECT** * **FROM** Instructor **WHERE** bonus **BETWEEN** 500 **AND** 1000;

(9) **SELECT** * **FROM** HR **WHERE** hireDate **BETWEEN** '01-MAY-2022' **AND** '31-MAY-2022';

(10) **SELECT** * **FROM** Instructor **WHERE** bonus **IN** (100, 200, 300);

An Overview of Simple Commands

(11) SELECT * FROM Instructor
WHERE position = 'assistant';

(12) SELECT * FROM R
WHERE lName > 'S';

(13) SELECT * FROM Instructor
WHERE fName **LIKE** 'J%';

(14) SELECT * FROM Instructor
WHERE fName **LIKE** 'J_h_';

(15) SELECT fName, lName FROM Instructor
WHERE address **LIKE** '**%**Houston,TX**%**';

(16) SELECT (salary+bonus) FROM Instructor;

(17) SELECT (salary+bonus) **AS** total_income FROM Instructor;

(18) SELECT ((A1/3.14)*2.54) **AS** A1_IN_CM FROM R;

(19)SELECT **SYSDATE** AS TODAY
FROM **DUAL**;

(20) **SELECT** Fname, Lname, Bday,
TRUNC (**MONTHS_BETWEEN** (**SYSDATE**, Bday)/12) **AS** "Actual Age"
FROM Person;

Join Operation

- JOIN is the relational operation that combines the data stored in two tables, creating a new result table
- JOIN is the most important operation in relational query processing

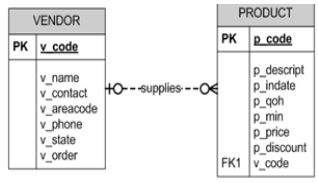
- **JOIN is performed between two tables that have a PK-FK relationship**.
  - If your FROM clause contains **2 tables**, you need **one join**.
  - If your FROM clause contains **3 tables**, you need **two joins**.
  - If your FROM clause contains ***N*** **tables**, you need ***N-1*** **joins**.
- **JOINing between two tables that do not have a PK-FK relationship may *NOT BE CORRECT*.**

JOIN: Two Syntax

JOIN is usually done through ***PK-FK chains***

- For each product, find their vendor code and names

```
SELECT    P.P_Code, V.V_Code, V.V_Name
FROM      Product P, Vendor V
WHERE     P.V_Code = V.V_code;
```




```
SELECT    P.P_Code, V.V_Code, V.V_Name
FROM      Product P INNER JOIN Vendor V
ON        P.V_Code = V.V_code;
```

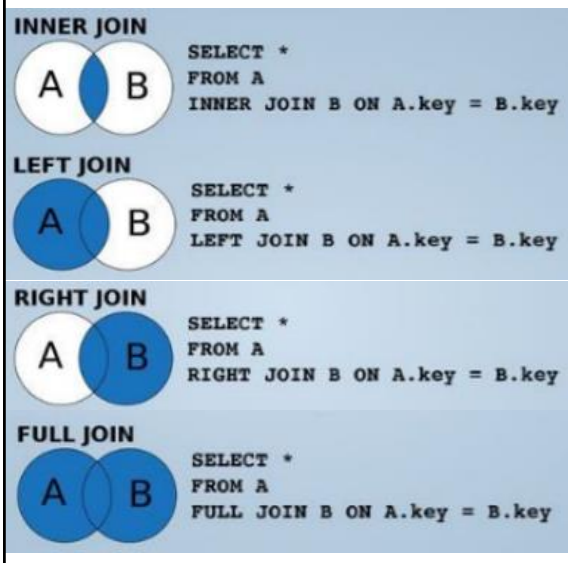

Types of JOINs

TABLE 8.1

SQL JOIN EXPRESSION STYLES

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
CROSS	CROSS JOIN	SELECT * FROM T1, T2	Returns the Cartesian product of T1 and T2 (old style)
		SELECT * FROM T1 CROSS JOIN T2	Returns the Cartesian product of T1 and T2
INNER	Old-style JOIN	SELECT * FROM T1, T2 WHERE T1.C1=T2.C1	Returns only the rows that meet the join condition in the WHERE clause (old style); only rows with matching values are selected
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns; the matching columns must have the same names and similar data types
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1=T2.C1	Returns only the rows that meet the join condition indicated in the ON clause
OUTER	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values

Types of JOINs



NATURAL JOIN:

SELECT *
FROM table_A
NATURAL JOIN table_B;

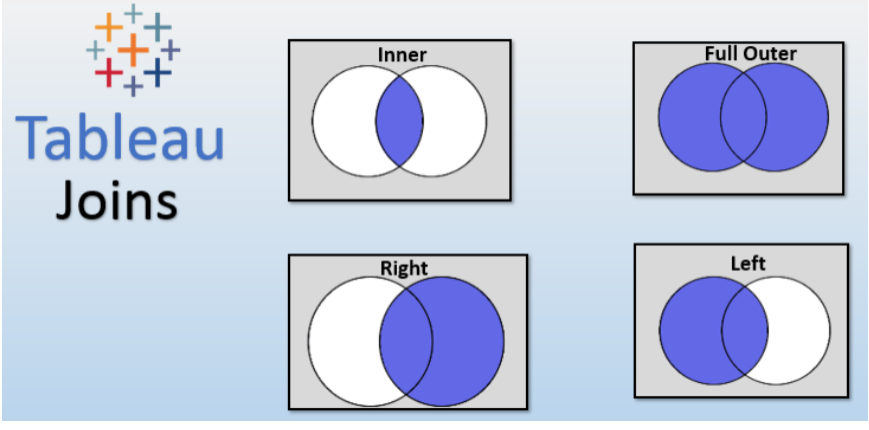
SELF JOIN:

SELECT *
FROM table_A X, table_A Y
WHERE X.A = Y.A;

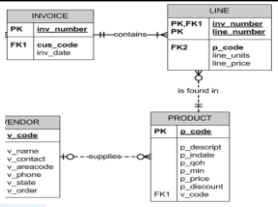
CROSS JOIN:

SELECT *
FROM table_A
CROSS JOIN table_B;

Types of JOINs in Tableau



INNER JOIN



• For each invoice, find the product description, #units, and price of items in the invoice.

SQL Plus

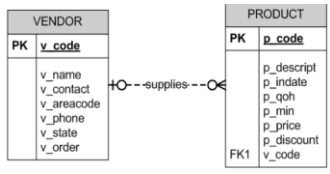
```
SQL> SELECT INVOICE.INV_NUMBER, PRODUCT.P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE
  2  FROM INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
  3       JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;

INV_NUMBER P_CODE     P_DESCRIPT                         LINE_UNITS LINE_PRICE
---------- ---------- ---------------------------------- ---------- ----------
      1001 13-Q2/P2   7.25-in. pwr. saw blade                    1      14.99
      1001 23109-HB   Claw hammer                                1       9.95
      1002 54778-2T   Rat-tail file, 1/8-in. fine               2       4.99
      1003 2238/QPD   B&D cordless drill, 1/2-in.                1      38.95
      1003 1546-QQ2   Hrd. cloth, 1/4-in., 2x50                  1      39.95
      1003 13-Q2/P2   7.25-in. pwr. saw blade                    5      14.99
      1004 54778-2T   Rat-tail file, 1/8-in. fine               3       4.99
      1004 23109-HB   Claw hammer                                2       9.95
      1005 PVC23DRT   PVC pipe, 3.5-in., 8-ft                   12       5.87
      1006 SM-18277   1.25-in. metal screw, 25                    3       6.99
      1006 2232/QTY   B&D jigsaw, 12-in. blade                    1     109.92
      1006 23109-HB   Claw hammer                                1       9.95
      1006 89-WRE-Q   Hicut chain saw, 16 in.                    1     256.99
      1007 13-Q2/P2   7.25-in. pwr. saw blade                    2      14.99
      1007 54778-2T   Rat-tail file, 1/8-in. fine               1       4.99
      1008 PVC23DRT   PVC pipe, 3.5-in., 8-ft                    5       5.87
      1008 WR3/TT3    Steel matting, 4'x8'x1/6", .5" mesh        3     119.95
      1008 23109-HB   Claw hammer                                1       9.95

18 rows selected.

SQL>
```


LEFT OUTER JOIN



•***For ALL the Vendors***, find all the products they provide. Include all the vendors whether or not they provide any product.

SQL Plus

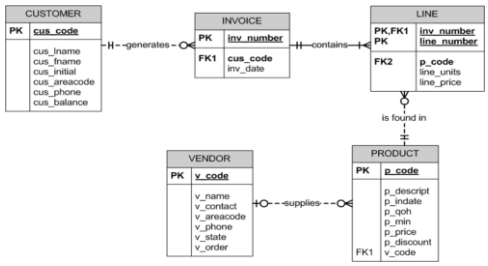
```
SQL> SELECT P_CODE, VENDOR.V_CODE, V_NAME
  2  FROM VENDOR LEFT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;

P_CODE     V_CODE V_NAME
---------- ------ ----------------------------------
11QER/31    25595 Rubicon Systems
13-Q2/P2    21344 Gomez Bros.
14-Q1/L3    21344 Gomez Bros.
1546-QQ2    23119 Randsets Ltd.
1558-QW1    23119 Randsets Ltd.
2232/QTY    24288 ORDVA, Inc.
2232/QWE    24288 ORDVA, Inc.
2238/QPD    25595 Rubicon Systems
23109-HB    21225 Bryson, Inc.
54778-2T    21344 Gomez Bros.
89-WRE-Q    24288 ORDVA, Inc.
SM-18277    21225 Bryson, Inc.
SW-23116    21231 D&E Supply
WR3/TT3     25595 Rubicon Systems
            22567 Dome Supply
            21226 SuperLoo, Inc.
            24004 Brackman Bros.
            25501 Damal Supplies
            25443 B&K, Inc.

19 rows selected.

SQL>
```

Multi-table joins



54

List customer last name, invoice number, invoice date, and product description for all invoices for customer 10014.

```
SELECT    CUS_LNAME, I.INV_NUMBER, INV_DATE, P_DESCRIPT
FROM      CUSTOMER C, INVOICE I, LINE L, PRODUCT P
WHERE     C.CUS_CODE = I.CUS_CODE
          AND  I.INV_NUMBER = L.INV_NUMBER
          AND  L.P_CODE = P.P_CODE
          AND  C.CUS_CODE = 10014
ORDER BY  INV_NUMBER;
```

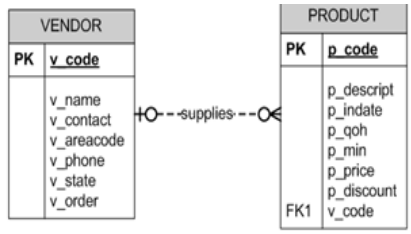

Subqueries

- Subquery (or nested query): a complete **SELECT** statement embedded within another **SELECT** statement.

```
SELECT    select_list
FROM      table
WHERE     expr operator
                (SELECT    select_list
                FROM       table);
```

- The results of this *inner* **SELECT** statement are used in the *outer* **SELECT** statement to help determine the final result.
- A subquery produces a *temporary table* with results
- Can be used in **SELECT, FROM, WHERE,** and **HAVING** clauses of an outer **SELECT** statement and also in **INSERT**, **UPDATE**, **DELETE** statements

Subqueries



- Find all vendor names which supplies USB

```
SELECT  v_name, v_phone
FROM    Vendor
WHERE   v_code IN (SELECT   v_code
                   FROM     Product
                   WHERE    p_descript = 'USB');
```

- Can we use "=" instead of "IN"?

```
SELECT  v_name, v_phone
FROM    Vendor
WHERE   v_code = (SELECT   v_code
                  FROM     Product
                  WHERE    p_descript = 'USB');
```


SQL Topics

- **SQL Basics**
  - Basic DDL and DML
  - Simple commands on a Single table
  - Joins (inner join, outer join, natural joins, multi-table joins)
  - Subquery
- **SQL for Data Engineering**
  - Date manipulation and Date Arithmetic
  - String manipulation
  - Merge
- **SQL for Analytics**
  - Aggregation, grouping, and ordering
  - OLAP functions
  - Ranking, partition by, buckets
  - Creating new derived columns

DATE Manipulation & Formats

- To insert a date using a default format:
  INSERT INTO Loan (Loan#, LoanDate) VALUES (12, '03-FEB-2019');
- To insert a date using a non-default format:
  INSERT INTO Loan (Loan#, LoanDate)
  VALUES (12, **TO_DATE**('06/02/2022', 'MM/DD/YYYY'));
- Show today's date in the form of MM/DD/YYYY format
  SELECT SYSDATE, **TO_CHAR**(SYSDATE, 'MM/DD/YYYY') as CurrDate FROM DUAL;
- Show today's date including time
  SELECT SYSDATE, **TO_CHAR**(SYSDATE, 'yyyy-mm-dd hh24:mi:ss') as CurrDateTime FROM DUAL;

DATE Arithmetic

- Find products whose INDATE is more than 90 days old

```
SELECT    P_Code, P_Desc, P_Indate
FROM      Product
WHERE     P_Indate <= SYSDATE - 90;
```

- Find all the orders received in Feb 2022.

```
SELECT Order#, Odate
FROM ORDER
WHERE Odate BETWEEN '01-Feb-22' AND '28-Feb-22';
```

DATE Extraction

- **EXTRACT** function: returns numerical value of YEAR, MONTH, or DTAE

Employee (**E#,** Ename, DoB)

-- Find the years of employees were born

```
SELECT    Ename, EXTRACT (YEAR FROM DoB)
FROM      Employee;
```

-- Month and day can also be extracted

```
SELECT Ename, EXTRACT (MONTH FROM DoB),
                      EXTRACT (DAY FROM DoB)
FROM Employee;
```


STRING Manipulation

- **Concatenating strings**
  SQL> SELECT Lname||', '||Fname "FULL NAME" FROM Emp;
  will display
  FULL NAME
  Bond, James

- **Handling Capitalization**
  SQL> SELECT Lname, **INITCAP**(Lname), **UPPER**(Lname), **LOWER**(Lname) FROM Emp;

- **Capitalization:** Show only the first letter in capital and the rest in lower
  SQL> SELECT **INITCAP**(**LOWER**(Lname)) FROM Emp;

- **Padding blanks to strings**
  RPAD: Pad to the right side of the column with left justification
  LPAD: Pad to the left side of the column with right justification
  SQL> SELECT **RPAD**(Lname, 15, '.'), Age, **LPAD**(Lname, 15), FROM EMP:

```
Clinton.....    53        ……..Clinton
Gore.....       52        ……..Gore
```

STRING Manipulation

/* **TRIM** function is used to remove all leading or trailing characters (or both) from a character string.*/

SQL>SELECT TRIM(“ LEADING Fellows “) FROM DUAL;

/* Use of **LTRIM**(left trim) and **RTRIM**(right trim) functions*/

SQL> SELECT Lname, **LTRIM**(Lname, 'SA'), **RTRIM**(Lname, 'S')
FROM Emp;

(LTRIM prints M from SAM, RTRIM prints WALE from WALES)

/* Remove **."** at the end and **"THE** from the front */

SQL> SELECT **LTRIM** (**RTRIM** (Title,'."'), ‘ “THE ') FROM Magazine;

Will convert

MY DARING."
"THE GOD FATHER."

into

MY DARING
GOD FATHER

STRING Manipulation

- **SUBSTR**
  - Syntax : SUBSTR(char, m, n)
  - Purpose : Returns a portion of the char, **beginning at character ‘m’ up to ‘n’** characters. If ‘n’ is omitted, result is returned up to the end of char.
  - Example : SELECT **SUBSTR**(first_name,1,3)
    FROM employee;

Output :

```
SUB
---
JOH
KEV
JEA
LYN
LES
CYN
```

STRING Manipulation

/* **SUBSTR**(String, starting_position, #chars to be returned) */

/* Extract numeric part, add 1000, and concatenate */

```
SQL> SELECT Lname, E_ID,
        'S' || TO_CHAR( TO_NUMBER (SUBSTR (E_ID,2,3) ) + 1000)
    "NEW E_#"
        FROM Emp;
```

Will display

LNAME	E_ID	NEW E_#
Jones	E001	S1001
Wales	E002	S1002

REPLACE() and TRANSLATE()

https://www.databasestar.com/oracle-translate/

TRANSLATE(string, from_string, to_string): From the ***string***, replace each character of ***from_string*** into the corresponding character of the ***to_string, char by char*** ***(case sensitive)***

REPLACE (string, from_string, to_string): From the ***string,*** replaces the ***entire string*** of ***from_string*** into ***to_string***, rather than one at a time like TRANSLATE.

```
SELECT 'Complete IT Professional' as SAMPLE_TEXT,
REPLACE('Complete IT Professional', 'let', 'sip') as REPLACE_TEST,
TRANSLATE('Complete IT Professional', 'let', 'sip') as TRANSLATE_TEST
FROM dual;
```

Result:

SAMPLE_TEXT	REPLACE_TEST	TRANSLATE_TEST
Complete IT Professional	Compsipe IT Professional	Compsipi IT Profissionas

MERGE

https://www.oracletutorial.com/oracle-basics/oracle-merge/

- A command for copying/merging data from a source table to a target table, utilizing matching conditions
- Template:

```
MERGE INTO target_table
USING source_table
ON search_condition
    WHEN MATCHED THEN
        UPDATE SET col1 = value1, col2 = value2,...
        WHERE <update_condition>
        [DELETE WHERE <delete_condition>]
    WHEN NOT MATCHED THEN
        INSERT (col1,col2,...)
        values(value1,value2,...)
        WHERE <insert_condition>;
```


MERGE

/* Update PRODUCT table with the data from NEW_PRODUCT
When their names matches, add the quantity; else insert a new row with the new product info */

```
MERGE INTO product prod
        USING new_product new_prod
        ON (prod.product_name = new_prod.product_name)
WHEN MATCHED THEN
        UPDATE SET prod.stock_qty = prod.stock_qty + new_prod.stock_qty
WHEN NOT MATCHED THEN
        INSERT (product_name, stock_qty)
        VALUES (new_prod.product_name, new_prod.qty);
```

MERGE

- Example source: https://www.oracletutorial.com/oracle-basics/oracle-merge/

/* Merge Member data to Member_Staging tables

- When member_ID matches, but if Fname, Lname, or Rank are different, then use member table data. When member_ID does not match, insert */

members

MEMBER_ID	FIRST_NAME	LAST_NAME	RANK
1	Abel	Wolf	Gold
2	Clarita	Franco	Platinum
3	Darryl	Giles	Silver
4	Dorthea	Suarez	Silver
5	Katrina	Wheeler	Silver
6	Lilian	Garza	Silver
7	Ossie	Summers	Gold
8	Paige	Mcfarland	Platinum
9	Ronna	Britt	Platinum
10	Tressie	Short	Bronze

member_staging

MEMBER_ID	FIRST_NAME	LAST_NAME	RANK
1	Abel	Wolf	Silver
2	Clarita	Franco	Platinum
3	Darryl	Giles	Bronze
4	Dorthea	Gate	Gold
5	Katrina	Wheeler	Silver
6	Lilian	Stark	Silver

MERGE

https://www.oracletutorial.com/oracle-basics/oracle-merge/

```
MERGE INTO member_staging x
USING members y
ON (x.member_id  = y.member_id)
WHEN MATCHED THEN
   UPDATE SET x.first_name = y.first_name,
              x.last_name = y.last_name,
              x.rank = y.rank
   WHERE x.first_name <> y.first_name OR
         x.last_name <> y.last_name OR
         x.rank <> y.rank
WHEN NOT MATCHED THEN
   INSERT(x.member_id, x.first_name, x.last_name, x.rank)
   VALUES(y.member_id, y.first_name, y.last_name, y.rank);
```

SQL Topics

- **SQL Basics**
  - Basic DDL and DML
  - Simple commands on a Single table
  - Joins (inner join, outer join, natural joins, multi-table joins)
  - Subquery
- **SQL for Data Engineering**
  - Date manipulation and Date Arithmetic
  - String manipulation
  - Merge
- **SQL for Analytics**
  - Aggregation, grouping, and ordering
  - OLAP functions
  - Ranking, partition by, buckets
  - Creating new derived columns

Analytic Functions in SQL

- SQL has many analytic functions:
  - Aggregation
  - OLAP (Online Analytical Functions)
  - Ranking
    - **RANK() & DENSE_RANK ()**
  - Ranking within subgroups
    - **PARTITION BY**
  - Creating new derived columns
    - **WIDTH_BUCKET ()**
    - **CASE…WHEN**
    - **COALESCE ()**
    - **LAG() and LEAD()**
    - **CUME_DIST()**

Aggregate Functions

- Aggregation functions *operate on a single column* of a table and ***return a single value***

- Five aggregate functions:
  - **COUNT**: returns the *number of values* in a specified column
  - **SUM**: returns the *sum of the values* in a specified column
  - **AVG**: returns the *average of the values* in a specified column
  - **MIN**: returns the *smallest value* in a specified column
  - **MAX**: returns the *largest value* in a specified column

- Where to use:
  - In the **SELECT** list
  - In the **HAVING** clause

Aggregation

- Example of five aggregation functions

```
SELECT MAX(bonus), MIN(bonus), AVG(bonus), COUNT(bonus),
SUM(bonus)
  FROM Instructor
  WHERE position = 'assistant';
```

- How many ***different*** course titles are there?

```
SELECT    COUNT(DISTINCT title)  AS  count
FROM      Course;
```

Aggregation

- Note: What's wrong with the following?

```
SELECT  instructorID, bonus
FROM    Instructor
WHERE   bonus > AVG(bonus);
```

Note: use aggregate functions only in **SELECT** and **HAVING** clause

- ***Why?***

- Rewrite using a subquery

```
SELECT  instructorID, bonus
FROM    Instructor
WHERE   bonus > (SELECT AVG(bonus)
                 FROM Instructor);
```


Descriptive Statistics in SQL

- **SELECT** cus_fname, cus_lname, cus_balance
  AVG(cus_balance) mean,
  MEDIAN (cus_balance) median,
  STATS_MODE (cus_balance) mode,
  STDDEV (cus_balance) "Standard Deviation"
  FROM Product;

CUSTOMER	
PK	**cus_code**
	cus_lname cus_fname cus_initial cus_areacode cus_phone cus_balance

- Computing median over partition

SELECT cus_fname, cus_lname, cus_balance
MEDIAN (cus_balance) **OVER** (**PARTITION BY** cus_areacode) AS median
FROM Product;

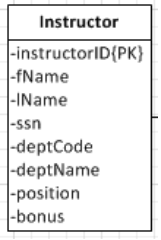
- Computing SD over unique values

SELECT cus_fname, cus_lname, cus_balance
STDDEV (**DISTINCT** cus_balance) **FROM** Product;

Aggregation with GROUP BY

Find the total number of instructors in each department and the sum of their bonus, respectively

```
SELECT    deptCode, COUNT(instructorID)  AS count, SUM(bonus) AS sum
FROM      Instructor
GROUP BY  deptCode
ORDER BY  deptCode;
```



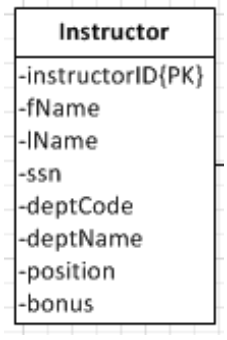
•Result:

deptCode	count	sum
acct	2	1100
math	2	300

•***GROUP BY*** *must include* ***all non-aggregate function column names*** *in the* ***SELECT*** *list.*

What's wrong with the following query?

Suppose we have the Instructor table as follows.



•Find the deptCode and deptName, and the total number of instructors in each department and the sum of their bonus, respectively

```
SELECT    deptCode, deptName,
          COUNT(instructorID)  AS count,
          SUM(bonus) AS sum
FROM      Instructor
GROUP BY  deptCode
ORDER BY  deptCode;
```

Aggregation with GROUP BY and HAVING

```
Oracle SQL*Plus
File  Edit  Search  Options  Help
SQL> SELECT V_CODE, COUNT(DISTINCT (P_CODE)), AVG(P_PRICE)
  2  FROM PRODUCT
  3  GROUP BY V_CODE;

   V_CODE COUNT(DISTINCT(P_CODE)) AVG(P_PRICE)
---------- ---------------------- ------------
     21225                      2         8.47
     21231                      1         8.45
     21344                      3        12.49
     23119                      2        41.97
     24288                      3   155.593333
     25595                      3        89.63
                                2       10.135

7 rows selected.

SQL> SELECT V_CODE, COUNT(DISTINCT (P_CODE)), AVG(P_PRICE)
  2  FROM PRODUCT
  3  GROUP BY V_CODE
  4  HAVING AVG(P_PRICE) < 10;

   V_CODE COUNT(DISTINCT(P_CODE)) AVG(P_PRICE)
---------- ---------------------- ------------
     21225                      2         8.47
     21231                      1         8.45
```

PRODUCT	
PK	p_code
	p_descript
	p_indate
	p_qoh
	p_min
	p_price
	p_discount
FK1	v_code

Aggregation with GROUP BY and HAVING

- Find the number of invoice per year and per month for the last 3 years

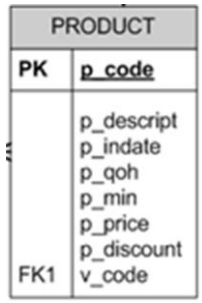
```
SELECT EXTRACT(year FROM inv_date) "Year",
       EXTRACT(month FROM inv_date) "Month",
       COUNT(inv_date) "No. of Invoices"
FROM invoice
GROUP BY EXTRACT(year FROM inv_date),
       EXTRACT(month FROM inv_date)
HAVING EXTRACT(year FROM inv_date) IN (2017, 2016, 2015)
ORDER BY "No. of Invoices" DESC;
```

INVOICE	
PK	inv_number
FK1	cus_code
	inv_date

Nested Subquery and Aggregate Functions

- Find the product that has the oldest date

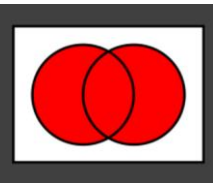
```
SELECT P_CODE, P_DESCRIPT
FROM   PRODUCT
WHERE  P_INDATE = (
    SELECT   MIN (P_INDATE)
    FROM     PRODUCT);
```



This query is in the form of a nested query.

SET Operations

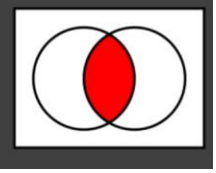
```
-- All customers and staff
SELECT first_name, last_name
FROM customer
UNION
SELECT first_name, last_name
FROM staff
```



UNION

$R \cup S$

```
-- customers that are actors
SELECT first_name, last_name
FROM customer
INTERSECT
SELECT first_name, last_name
FROM actor
```

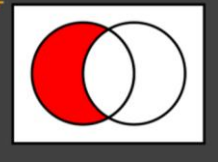


INTERSECT

$R \cap S$

MINUS

```
-- customers that are not staff
SELECT first_name, last_name
FROM customer
EXCEPT
SELECT first_name, last_name
FROM staff
```



MINUS

$R - S$

SQL OLAP Functions: ROLLUP

- Example: **GROUP BY ROLLUP** (V_Code, P_Code) produces the union of
  - GROUP BY V_Code, P_Code
  - GROUP BY V_Code
  - Grand total

```
SQL> SELECT   V_CODE, P_CODE, SUM(SALE_UNITS*SALE_PRICE) AS TOTSALES
  2  FROM     DWDAYSALESFACT NATURAL JOIN DWPRODUCT NATURAL JOIN DWVENDOR
  3  GROUP BY ROLLUP (V_CODE, P_CODE)
  4  ORDER BY V_CODE, P_CODE;
```


59

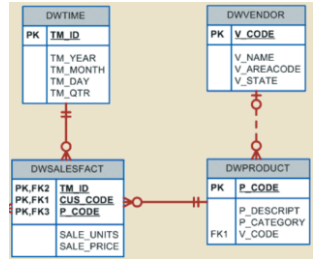
Oracle SQL*Plus

File Edit Search Options Help

```
SQL> SELECT   V_CODE, P_CODE, SUM(SALE_UNITS*SALE_PRICE) AS TOTSALES
  2  FROM     DWDAYSALESFACT NATURAL JOIN DWPRODUCT NATURAL JOIN DWVENDOR
  3  GROUP BY ROLLUP (V_CODE, P_CODE)
  4  ORDER BY V_CODE, P_CODE;

    V_CODE P_CODE       TOTSALES
---------- ---------- ----------
     21225 23109-HB          99.5
     21225 PVC23DRT        199.58
     21225 SM-18277         41.94
     21225                 341.02
     21344 13-Q2/P2        239.84
     21344 54778-2T         59.88
     21344                 299.72
     23119 1546-QQ2          79.9
     23119                   79.9
     24288 2232/QTY        219.84
     24288 89-WRE-Q        513.98
     24288                 733.82
     25595 2238/QPD          77.9
     25595 WR3/TT3          719.7
     25595                 797.6
                          2252.06

16 rows selected.
```



Subtotals by V_CODE

Grand total for all P_CODE values

60

SQL OLAP Functions: Cube

- The CUBE extension
  - Enables you to get a subtotal for each column listed in the expression, in addition to a grand total for the last column listed
  - GROUP BY CUBE (TM_Month, P_CODE) is a union of:
    - GROUP BY (TM_Month, P_CODE)
    - GROUP BY (TM_Month)
    - GROUP BY (P_CODE)
    - Grand total

```
SQL> SELECT   TM_MONTH, P_CODE, SUM(SALE_UNITS*SALE_PRICE) AS TOTSALES
  2  FROM     DWDAYSALESFACT NATURAL JOIN DWPRODUCT NATURAL JOIN DWTIME
  3  GROUP BY CUBE (TM_MONTH, P_CODE)
  4  ORDER BY TM_MONTH, P_CODE;
```

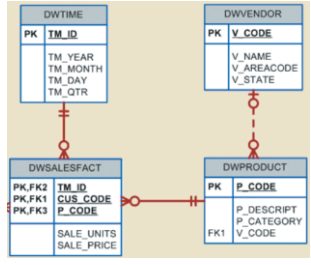

Oracle SQL*Plus

File Edit Search Options Help

```
SQL> SELECT   TM_MONTH, P_CODE, SUM(SALE_UNITS*SALE_PRICE) AS TOTSALES
  2  FROM     DWDAYSALESFACT NATURAL JOIN DWPRODUCT NATURAL JOIN DWTIME
  3  GROUP BY CUBE (TM_MONTH, P_CODE)
  4  ORDER BY TM_MONTH, P_CODE;

 TM_MONTH P_CODE       TOTSALES
--------- ---------- ----------
        9 13-Q2/P2         134.91
        9 1546-QQ2          79.9
        9 2232/QTY        109.92
        9 2238/QPD          77.9
        9 23109-HB          59.7
        9 54778-2T         39.92
        9 89-WRE-Q        256.99
        9 PUC23DRT         99.79
        9 SM-18277         20.97
        9 WR3/TT3         359.85
        9                1239.85
       10 13-Q2/P2        104.93
       10 2232/QTY        109.92
       10 23109-HB          39.8
       10 54778-2T         19.96
       10 89-WRE-Q        256.99
       10 PUC23DRT         99.79
       10 SM-18277         20.97
       10 WR3/TT3         359.85
       10                1012.21
          13-Q2/P2        239.84
          1546-QQ2          79.9
          2232/QTY        219.84
          2238/QPD          77.9
          23109-HB          99.5
          54778-2T         59.88
          89-WRE-Q        513.98
          PUC23DRT        199.58
          SM-18277         41.94
          WR3/TT3          719.7
                          2252.06

31 rows selected.
```



Subtotals by month

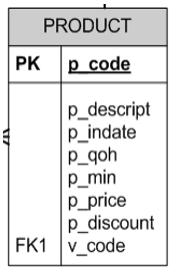
Subtotals by product

Grand total for all products and months

RANK and DENSE RANK Functions

- RANK() leave ranking gaps when there are multiple rows in the same rank (1, 2, 2, 4, 5…)
- DENSE_RANK() does **not** leave ranking gaps (1, 2, 2, 3, 4, 5)

```
SELECT p_code, p_descript, p_price,
        RANK() OVER
                (ORDER BY p_price NULLS LAST) AS Rank,
        DENSE_RANK ()
                (ORDER BY p_price NULLS LAST) AS Dense_rank
FROM    Product
ORDER BY p_price;
```

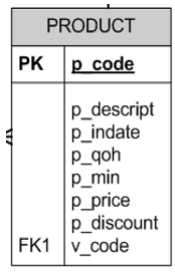


- DESC means descending order
- NULLS LAST means null values are smaller than non-null values
- You may use NULLS FIRST

RANK and DENSE RANK Functions

- Ranking applied to dates

```
SELECT p_code, p_descript, p_indate,
        RANK() OVER
                (ORDER BY p_indate NULLS LAST) AS Rank,
        DENSE_RANK ()
                (ORDER BY p_indate NULLS LAST) AS Dense_rank
FROM    Product
ORDER BY p_indate;
```



Top-N Queries

- **In Oracle 12c and after:**

```
SELECT      p_code, p_descript, p_price,
        RANK() OVER
                (ORDER BY p_price NULLS LAST) AS Rank,
FROM Product
ORDER BY p_price;
FETCH FIRST 10 ROWS ONLY;
```

- **Another option by percentage**

FETCH FIRST 20 PERCENT ROWS ONLY;

- **MySQL uses LIMIT clause**

LIMIT 10

PARTITION BY

- Partition the records by a column and them apply the function
- Ex: For each employee, show his/her dept# and the total# of employees working for the dept.

```
SELECT empno, deptno, COUNT(*)
        OVER (PARTITION BY deptno) DEPT_COUNT
FROM emp;
```

```
emp_no  dept_no  DEPT_COUNT
1       10       3
2       10       3
3       10       3 <- three because there are three "dept_no = 10" records
4       20       2
5       20       2 <- two because there are two "dept_no = 20" records
```

https://stackoverflow.com/questions/561836/oracle-partition-by-keyword

RANK with PARTITION BY

- Create ranks within each partition

```
SELECT employee_id, first_name, department_id, salary
RANK() OVER (PARTITION BY department_id
        ORDER BY salary) rank
FROM employee;
```

	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	RANK
1	119	Karen	30	2500	1
2	118	Guy	30	2600	2
3	117	Sigal	30	2800	3
4	116	Shelli	30	2900	4
5	115	Alexander	30	3100	5
6	114	Den	30	11000	6
7	144	Peter	50	2500	1
8	143	Randall	50	2600	2
9	142	Curtis	50	3100	3
10	141	Trenna	50	3500	4
11	107	Diana	60	4200	1
12	105	David	60	4800	2

Windows Function: WIDTH_BUCKET

- This operation assigns a bucket number to the expression that it evaluates. It helps in generating equi-width histograms.
- Example: Show the last names, salaries and bucket numbers of all employees according to the following rule:
  - Salary <30,000 -> 0
  - 30,000-40,000 -> 1
  - 40,000-50,000 -> 2
  - Salary >50,000 -> 3

```
SQL> SELECT lname, salary,
WIDTH_BUCKET(salary, 30000, 50000, 2)   AS Bucket
FROM employee;
```

LNAME	SALARY	BUCKET
Smith	30000	1
Wong	40000	2
Zelaya	25000	0
Wallace	43000	2
Narayan	38000	1
English	25000	0
Jabbar	25000	0
Borg	55000	3

8 rows selected.

CASE Statement

- **Simple CASE Statements**

```
SELECT      Fname, Lname,
      (CASE DNO
            WHEN 1      THEN 'Headquarters'
            WHEN 4      THEN 'Administration'
            WHEN 5      THEN 'Research'
            ELSE         'No department'
      END)  AS Department
      FROM  Employee;
```

Output:

Fname	Lname	Department
John	Smith	Research
Franklin	Wong	Research
Alica	Zelaya	Administration

CASE Statement

- **Searched CASE Statements**

```
SELECT      Fname, Lname, Salary
      (CASE Salary
            WHEN  Salary <= 25000     THEN 1500
            WHEN  Salary > 25000 AND Salary < 50000    THEN 1000
            WHEN  Salary > 50000 AND Salary < 100000 THEN 500
            ELSE              0
      END)  "Bonus"
      FROM  Employee;
```

Output:

Fname	Lname	Salary	Bonus
John	Smith	30000	1000
Franklin	Wong	40000	1000
Alica	Zelaya	25000	1500

NULL and NVL Function

- Changing the display of null values
  SQL> SET NULL N/A
  All null values are displayed as N/A.
  SQL> SET NULL -- to reset the null to blank

- Using ***NVL*** function to change the null value to another value (Only in Oracle)
  SQL> SELECT Fname, Lname, NVL(CreditCode, 999) FROM Cust;

  - If the value of CreditCode is not null, the value is displayed
  - If the value of CreditCode is null, 999 is displayed.
  - NVL(v1,v2): *v1* and *v2* must match by their data types.
  - But *v1* can be any data type.

NULL and NVL Function

- Changing the display of null values
  SQL> SET NULL N/A
  All null values are displayed as N/A.
  SQL> SET NULL -- to reset the null to blank

- Using ***NVL*** function to change the null value to another value (Only in Oracle)
  SQL> SELECT Fname, Lname, NVL(CreditCode, 999) FROM Cust;

  - If the value of CreditCode is not null, the value is displayed
  - If the value of CreditCode is null, 999 is displayed.
  - NVL(v1,v2): *v1* and *v2* must match by their data types.
  - But *v1* can be any data type.

COALESCE Function

- A general form of NVL function-SQL 92 standard
- Provides a default value in case the column returns a NULL.
- ```
  SELECT
     S.storecode,
     coalesce(s.date, '2021-05-10') as date,
     coalesce(s.salesqty,0) as salesqty,
     coalesce(s.salesrevenue, 0)as salesrevenue
  FROM Products P;
  ```

- NVL uses only 2 arguments, while COALESCE can have multiple arguments, but stops at the first NOT NULL value
- Example:

Gives a 10% discount to all products with a list price. If there is no list price, then the sale price is the minimum price. If there is no minimum price, then the sale price is "5":

COALESCE Function

- Gives a 10% discount to all products with a list price. If there is no list price, then the sale price is the minimum price. If there is no minimum price, then the sale price is "5":

```
SELECT product_id, list_price, min_price,
   COALESCE(0.9*list_price, min_price, 5) Sale
   FROM product_information
   WHERE supplier_id = 102050
   ORDER BY product_id, list_price, min_price, Sale;
```

PRODUCT_ID	LIST_PRICE	MIN_PRICE	Sale
1769	48		43.2
1770		73	73
2378	305	247	274.5
2382	850	731	765
3355			5
```

LAG(), LEAD()

- Lead() allows you to grab data from ***a next row*** without having to do a self join and put it into the current row
- Lag() allows you to grab data from ***a previous row*** and put it into the current row
- Syntax:

```
LEAD(scalar_expression ,offset [,default])
OVER (
      [PARTITION BY partition_expression, … ]
      ORDER BY sort_expression [ASC | DESC], …
```


LAG(), LEAD()

- Display employee joinDate and EndingDate in one row

```
SELECT empcode, empname, joindate,
       LEAD (joinDate,1) OVER (ORDER BY  joinDate ASC)
      as EndDate
FROM employee;
```

EMPCODE	EMPNAME	JOININGDATE
1	Rajendra	01-SEP-18
2	Manoj	01-OCT-18
3	Sonu	10-MAR-18
4	Kashish	25-OCT-18
5	Tim	01-DEC-18
6	Akshita	01-NOV-18

EMPCODE	EMPNAME	JOININGDATE	ENDDATE
3	Sonu	10-MAR-18	01-SEP-18
1	Rajendra	01-SEP-18	01-OCT-18
2	Manoj	01-OCT-18	25-OCT-18
4	Kashish	25-OCT-18	01-NOV-18
6	Akshita	01-NOV-18	01-DEC-18
5	Tim	01-DEC-18	- NULL

LAG(), LEAD()

- Given sales data below, display year, sales amount, and increase from the previous year

```
SELECT year, sales,
    Sale –LAG (sale, 1, NULL) OVER (ORDER BY year
ASC) AS increase-from-last-year
FROM sales_table;
```

year	sale
2017	100
2018	125
2019	140
2020	175

year	sale	increase-from-last-year
2017	100	NULL
2018	125	25
2019	140	15
2020	175	35

CUME_DIST()

- CUME_DIST() find the cumulative distribution of a value within a group of values, between 0 and 1.
  - Represents the number of rows that have values less than or equal to that row's value ***divided by*** the total number of rows

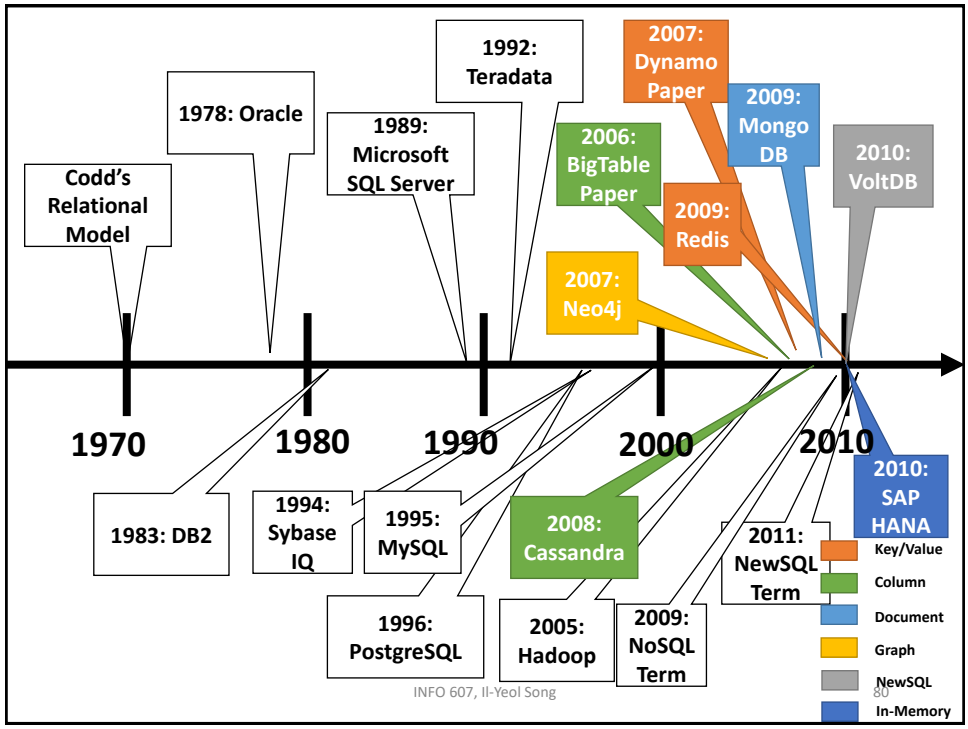
```
SELECT first_name, last_name, fees_paid,
CUME_DIST() OVER (ORDER BY fees_paid) AS cume_dist_test
FROM student;
```

FIRST_NAME	LAST_NAME	FEES_PAID	CUME_DIST_TEST
Julie	Armstrong	0	0.07142857143
Mark	Anderson	45	0.1428571429
Steven	Webber	80	0.2142857143
John	Smith	100	0.4285714286
Robert	Pickering	100	0.4285714286
Mary	Taylor	100	0.4285714286
Susan	Johnson	150	0.5714285714
Tanya	Hall	150	0.5714285714
Jarrad	Winston	300	0.6428571429
Tom	Capper	320	0.7142857143
Andrew	Cooper	400	0.7857142857
Mark	Holloway	410	0.8571428571
John	Rogers	700	0.9285714286
Michelle	Randall	(null)	1

Other Advanced SQL

- Other SQL functions and commands
- Views and Materialized views
- Indexes
- SQL with JSON
- SQL with Python
- PL/SQL
  - Stored functions
  - Stored Procedures
  - Triggers
  - Packages

79



80

Major Features of NoSQL Databases (1)

- Manage ***non-relational* data**: Semi and non-structured data
- Stores ***aggregated objects***:
  - Stores all the related data together (denormalzied objects)
    - No complex relationships
  - Could cause redundant data and update anomaly
  - Write-once-read-many applications
- ***Schema-on-read, mostly***
  - Some column stores need to define the schema in advance, but columns can be added dynamically
- Built on ***the scale-out architecture***
  - Parallel processing, Scalability, fault tolerance, High availability (HA)
  - Support ***automatic sharding***
    - Data are automatically distributed to nodes based on some fields

Major Features of NoSQL Databases (2)

- Soften the ACID property:
  - **BASE**: *Basically **Available**, Soft state, **Eventual consistency***
    - ***NoSQLs focuse on scalability and availability***
  - The data (and its replicas) will become consistent at some point in time after each transaction
  - Supports high throughput and low latency
- Open-sourced
- Four types
  - Key-value stores, Wide Column stores, Document stores
  - Graph databases
- Designed for **real-time OLTP system for non-uniform, big data**.

Rankings of DB Systems by Popularity

- NoSQL is moving up.
  - DB-Engines Ranking *by popularity* (https://db-engines.com/en/ranking May, 2022): From Websites, Google trends, social networks, job Ads, etc.

Rank May 2022	Rank Apr 2022	Rank May 2021	DBMS	Database Model
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	↑ 7.	Redis	Key-value, Multi-model
7.	↑ 8.	↓ 6.	IBM Db2	Relational, Multi-model
8.	↓ 7.	8.	Elasticsearch	Search engine, Multi-model
9.	9.	↑ 10.	Microsoft Access	Relational
10.	10.	↓ 9.	SQLite	Relational
11.	11.	11.	Cassandra	Wide column
12.	12.	12.	MariaDB	Relational, Multi-model
13.	13.	13.	Splunk	Search engine
14.	14.	↑ 27.	Snowflake	Relational
15.	15.	15.	Microsoft Azure SQL Database	Relational, Multi-model
16.	16.	16.	Amazon DynamoDB	Multi-model
17.	17.	↓ 14.	Hive	Relational
18.	18.	↓ 17.	Teradata	Relational, Multi-model
19.	19.	19.	Neo4j	Graph

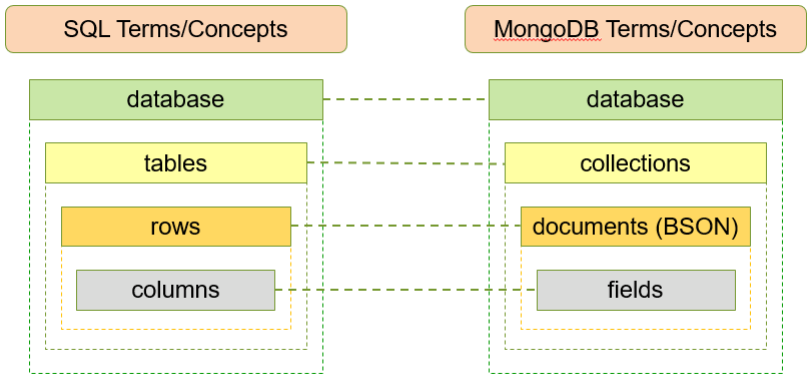
MongoDB Shell

The following Mongo Shell provides a cloud based MongoDB engine. You can create database, insert/update records, queries, etc.

https://docs.mongodb.com/manual/tutorial/query-embedded-documents/

- The shell resets all data stored once closed.
- Create commands in Notepad and then copy and paste into the shell
- Be careful about quotes
- Up arrow display the previous command

RDBMS and MongoDB



85

JSON

- JavaScript Object Notation (JSON) – data interchange format used to represent data as a logical object.
- Every JSON document requires an object ID.
- **Object** is enclosed by a pair of **curly brackets** {**K**:**V**}
- **Array** is enclosed by a pair of **square brackets [ ]**
- **Value could be a "string", an object, an array of values, a set of objects.**
- Example:

```
{
_id: 101,
title: "Database Systems",
author: ["Coronel", "Morris"],
publisher: {
        name: "Cengage",
        street: "500 Topbooks Avenue",
        city: "Boston",
        state: "MA"
        }
}
```


86

CRUD Example

```
> db.user.insertOne({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{
        "_id" : ObjectId("51…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.updateOne(
        {"_id" :
ObjectId("51…")},
        { $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```


CRUD: Creating/Inserting documents

- Example:- **db.<<collectionName>>.insertOne({document})**

```
db.users.insertOne(          <-- collection
   {
      name: "sue",           <-- field: value
      age: 26,               <-- field: value
      status: "pending"      <-- field: value
   }                                  } document
)
```

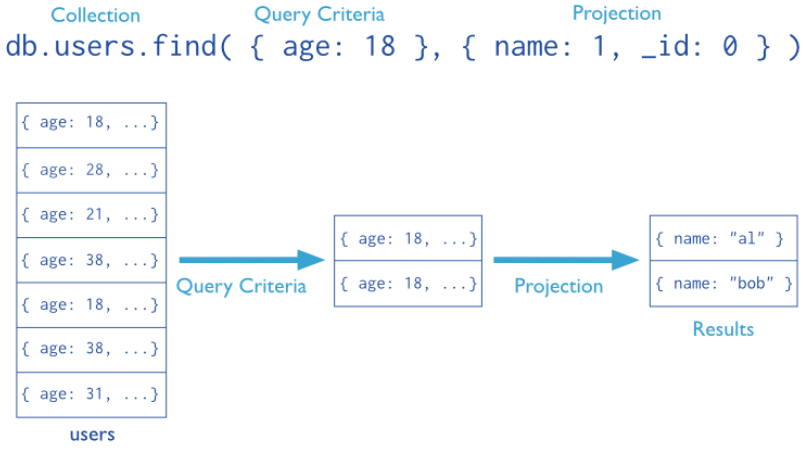
```
>>> db.emp.insert({id: 1, name: "Song"})
WriteResult({ "nInserted" : 1 })
>>>
```

CRUD: Inserting Multiple documents

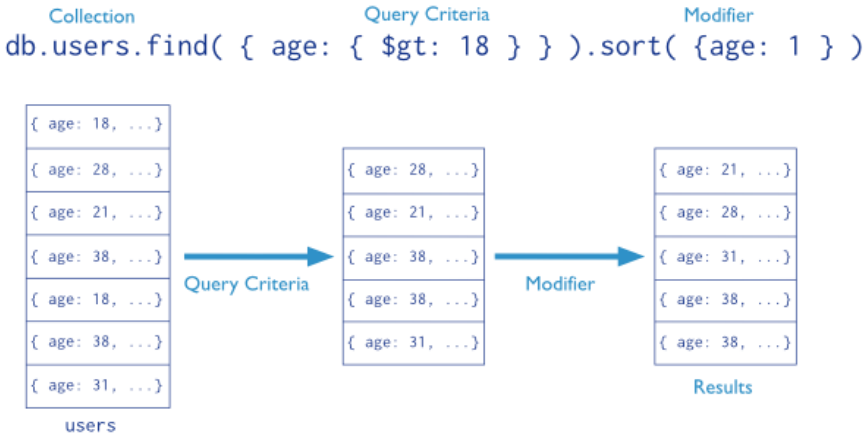
- Use the method – **insertMany([{d1}, {d2}.., {dn}])**
- Example:-

```
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
  { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
  { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```


CRUD: Find() with Query and Projections



CRUD: find() with Query Criteria



CRUD: UPDATE

- Update ***all the documents*** whose age is greater than 25

```
UPDATE people
SET status = "C"
WHERE age > 25
```

```
db.people.updateMany(
  { age: { $gt: 25 } },
  { $set: { status: "C" } }
)
```

- Update ***the first document*** whose age is greater than 25

```
db.people.updateOne(
  { age: { $gt: 25 } },
  { $set: { status: "C" } }
)
```

CRUD: DELETE

```
DELETE FROM users
WHERE status = "D"
```

- Remove ***the first*** document from the users collection where the status field equals "D":

  db.users.deleteOne({ status: "D" })

- Remove ***all the documents*** from the users collection where the status field equals "D":
  db.users.deleteMany({ status : "D" })

- Remove ***all the documents*** from the users collection:
  db.users.deleteMany({ })

MongoDB: Other Topics

- Pattern Matching
- Array manipulation
- Aggregation framework
- Indexing
- Document design
- PyMongo
- Architecture
- Replication
- Sharding

Good Resources

- **SQL Tutorial - Full Database Course for Beginners**, freeCodeCamp.org, https://www.youtube.com/watch?v=HXV3zeQKqGY
- **SQL Tutorial**, https://www.w3schools.com/sql/

- **Mongo DB Basics, Jyoti Reddy, Data Engineer at Krones**
- https://www.kdnuggets.com/2019/06/mongo-db-basics.html
- **MongoDB Tutorial for Beginners: Learn Basics in 7 Days,** David TaylorUpdatedApril 23, 2022. https://www.guru99.com/mongodb-tutorials.html

Question?