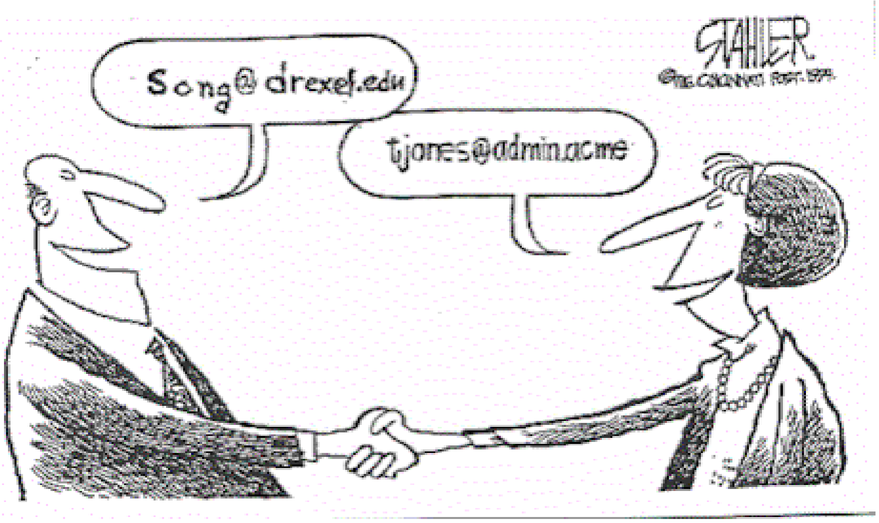
# LEADS 3-Day Camp
# Session 2: Big Data Management: Relational Databases and SQL

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/

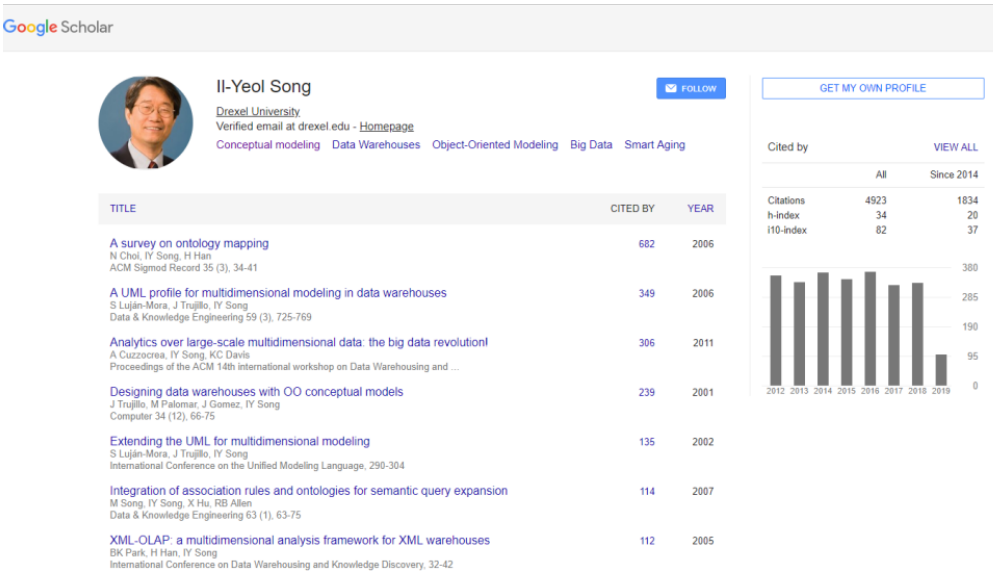
## Hello!



# Instructor

- Professor, College of Computing & Informatics, Drexel University,
- PhD in CS, LSU, Baton Rouge, USA, 1988
- 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 Journal
- Co-Chair, **iSchool Model Data Science Curriculum Committee**
- **Chair, IEEE Big Data and Smart Computing (BigComp) Conference**
- Published about 200+ papers

# My Google Scholar Page



Il-Yeol Song

Drexel University

Verified email at drexel.edu - Homepage

Conceptual modeling Data Warehouses Object-Oriented Modeling Big Data Smart Aging

| TITLE | CITED BY | YEAR |
|---|---|---|
| A survey on ontology mapping<br>N Choi, IY Song, H Han<br>ACM Sigmod Record 35 (3), 34-41 | 682 | 2006 |
| A UML profile for multidimensional modeling in data warehouses<br>S Luján-Mora, J Trujillo, IY Song<br>Data & Knowledge Engineering 59 (3), 725-769 | 349 | 2006 |
| Analytics over large-scale multidimensional data: the big data revolution!<br>A Cuzzocrea, IY Song, KC Davis<br>Proceedings of the ACM 14th international workshop on Data Warehousing and ... | 306 | 2011 |
| Designing data warehouses with OO conceptual models<br>J Trujillo, M Palomar, J Gomez, IY Song<br>Computer 34 (12), 66-75 | 239 | 2001 |
| Extending the UML for multidimensional modeling<br>S Luján-Mora, J Trujillo, IY Song<br>International Conference on the Unified Modeling Language, 290-304 | 135 | 2002 |
| Integration of association rules and ontologies for semantic query expansion<br>M Song, IY Song, X Hu, RB Allen<br>Data & Knowledge Engineering 63 (1), 63-75 | 114 | 2007 |
| XML-OLAP: a multidimensional analysis framework for XML warehouses<br>BK Park, H Han, IY Song<br>International Conference on Data Warehousing and Knowledge Discovery, 32-42 | 112 | 2005 |

Cited by

|  | All | Since 2014 |
|---|---|---|
| Citations | 4923 | 1834 |
| h-index | 34 | 20 |
| i10-index | 82 | 37 |

## Quick Survey

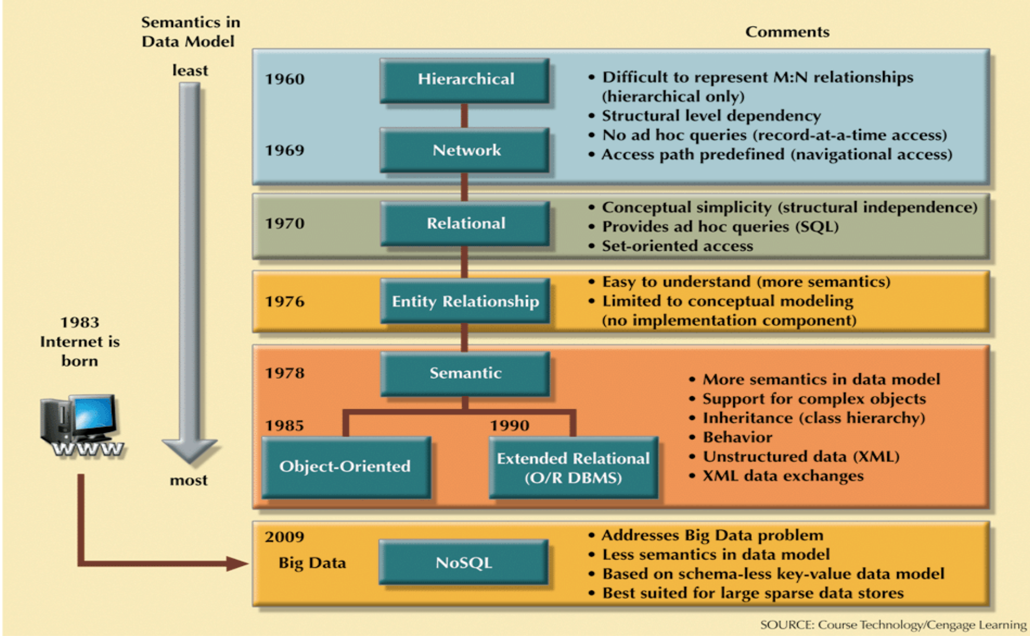
- How many of you are familiar with:
  - Relational database concepts?
  - SQL?
  - Any experience of using relational databases (Oracle, MS SQL Server, MySQL, Postgres, DB2, etc)?
  - Entity-relationship Modeling?
  - Understand Difference between RDBs and NoSQL databases?
  - Used MongoDB?

## Acknowledgement

- Materials 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*, 12th Edition, 2016. Cengage Learning.

FIGURE 2.6 The evolution of data models



Semantics in Data Model

least

most

1983 Internet is born

Comments

1960 Hierarchical

1969 Network

- Difficult to represent M:N relationships (hierarchical only)
- Structural level dependency
- No ad hoc queries (record-at-a-time access)
- Access path predefined (navigational access)

1970 Relational

- Conceptual simplicity (structural independence)
- Provides ad hoc queries (SQL)
- Set-oriented access

1976 Entity Relationship

- Easy to understand (more semantics)
- Limited to conceptual modeling (no implementation component)

1978 Semantic

1985 Object-Oriented

1990 Extended Relational (O/R DBMS)

- More semantics in data model
- Support for complex objects
- Inheritance (class hierarchy)
- Behavior
- Unstructured data (XML)
- XML data exchanges

2009 Big Data NoSQL

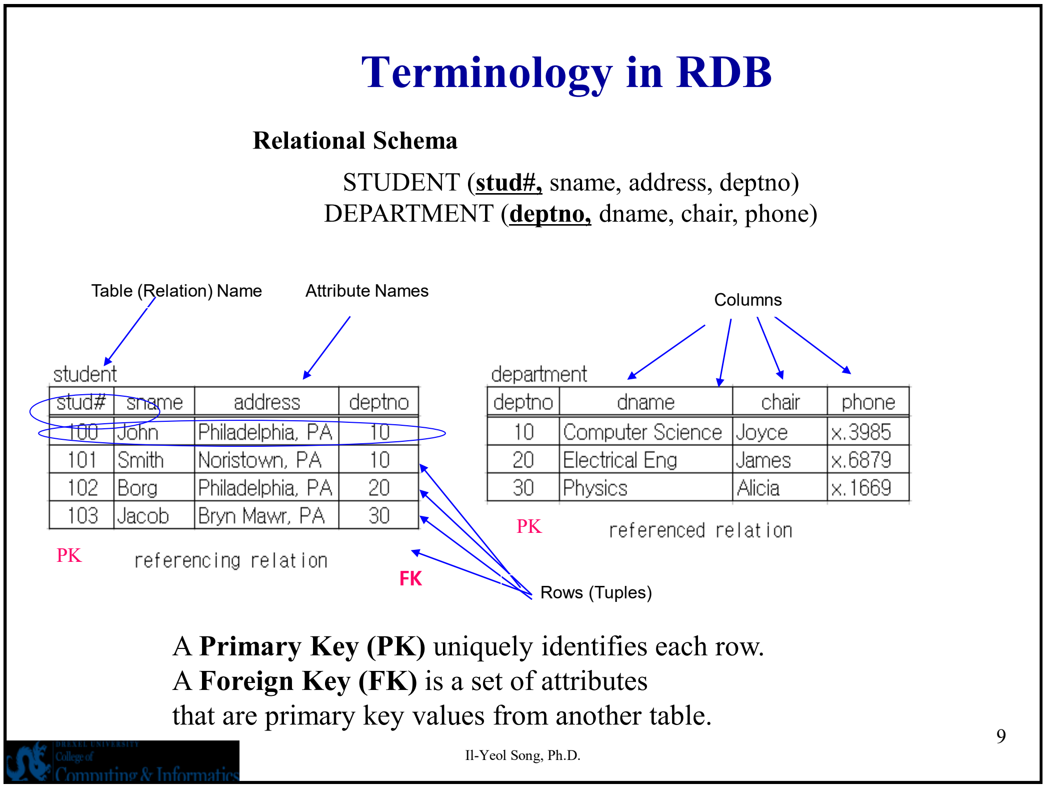
- Addresses Big Data problem
- Less semantics in data model
- Based on schema-less key-value data model
- Best suited for large sparse data stores

SOURCE: Course Technology/Cengage Learning

*(Coronel & Morris, 2016)*

# 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
- Null values
- 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
- Why relational databases have been popular in industry?



# Terminology in RDB

**Relational Schema**

STUDENT (<u>**stud#,**</u> sname, address, deptno)
DEPARTMENT (<u>**deptno,**</u> dname, chair, phone)

Table (Relation) Name

Attribute Names

Columns

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 |

PK

referencing relation

FK

department

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

PK

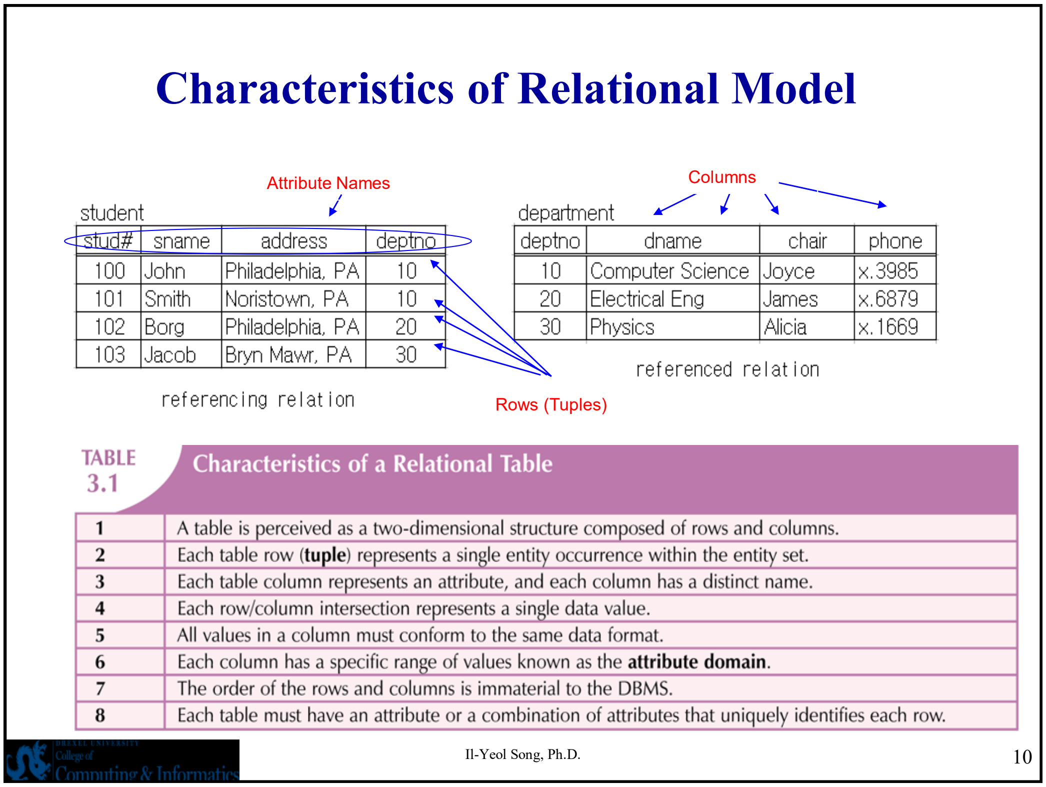
referenced relation

Rows (Tuples)

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.

9

Il-Yeol Song, Ph.D.



# Characteristics of Relational Model

Attribute Names

Columns

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

Rows (Tuples)

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. |

Il-Yeol Song, Ph.D.

10

## 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

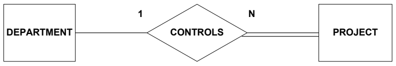
| deptno | dname | chair | phone |
|---|---|---|---|
| 10 | Computer Science | Joyce | x.3985 |
| 20 | Electrical Eng | James | x.6879 |
| 30 | Physics | Alicia | x.1669 |

referenced relation

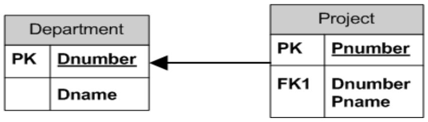
```
CREATE TABLE Department (
          DeptNo              NUMBER(5)      PRIMARY KEY,
          DName               VARCHAR2(15) NOT NULL      ,
          Chair               VARCHAR2 (20)              ,
          Phone               CHAR (10)                  ,
          );

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)
          );
```

# ER Model: Popular Notations

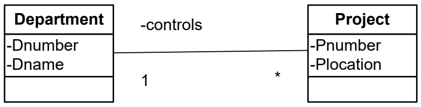
Chen



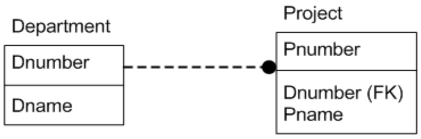
Visio 's Relational notation



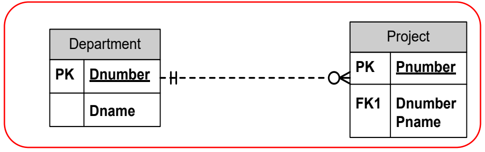
UML



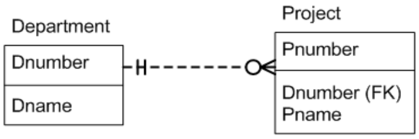
IDEF1X



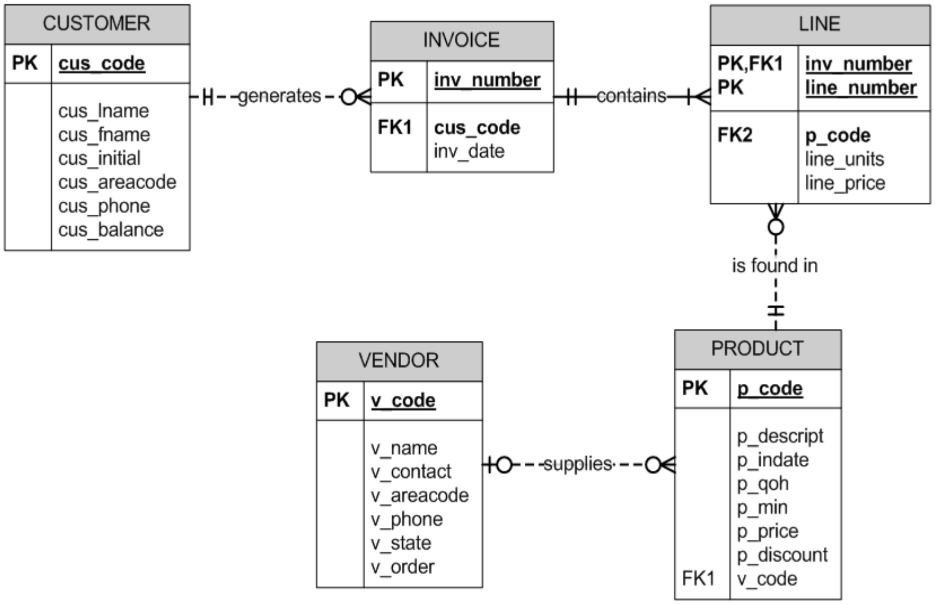
Crow's foot with Relational Notation in Visio



Visio IDEF1X with Crow's foot



# ERD



| CUSTOMER | |
|---|---|
| PK | cus_code |
| | cus_lname<br>cus_fname<br>cus_initial<br>cus_areacode<br>cus_phone<br>cus_balance |

| INVOICE | |
|---|---|
| PK | inv_number |
| FK1 | cus_code<br>inv_date |

| LINE | |
|---|---|
| PK,FK1<br>PK | inv_number<br>line_number |
| FK2 | p_code<br>line_units<br>line_price |

| VENDOR | |
|---|---|
| PK | v_code |
| | v_name<br>v_contact<br>v_areacode<br>v_phone<br>v_state<br>v_order |

| PRODUCT | |
|---|---|
| PK | p_code |
| <br><br><br><br><br><br>FK1 | p_descript<br>p_indate<br>p_qoh<br>p_min<br>p_price<br>p_discount<br>v_code |

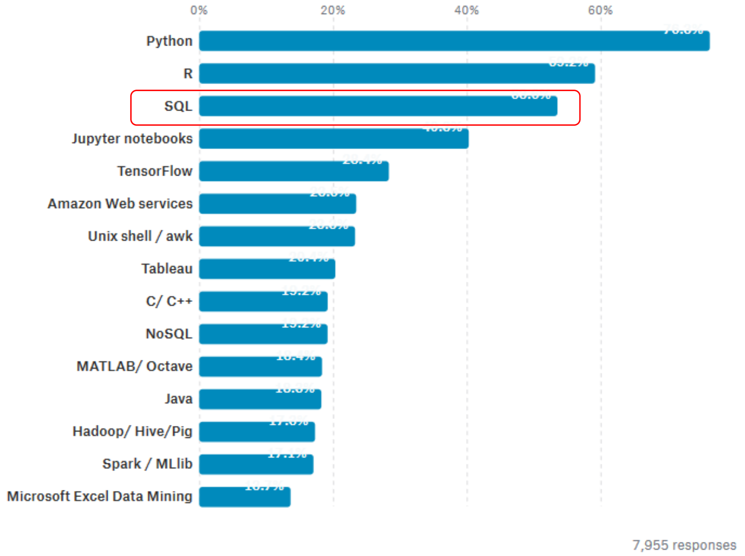
(Coronel & Morris, 2016)

# 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

## Top Tools used in Data Science (2017)



Source: https://www.kaggle.com/surveys/2017

## Two major Components of SQL

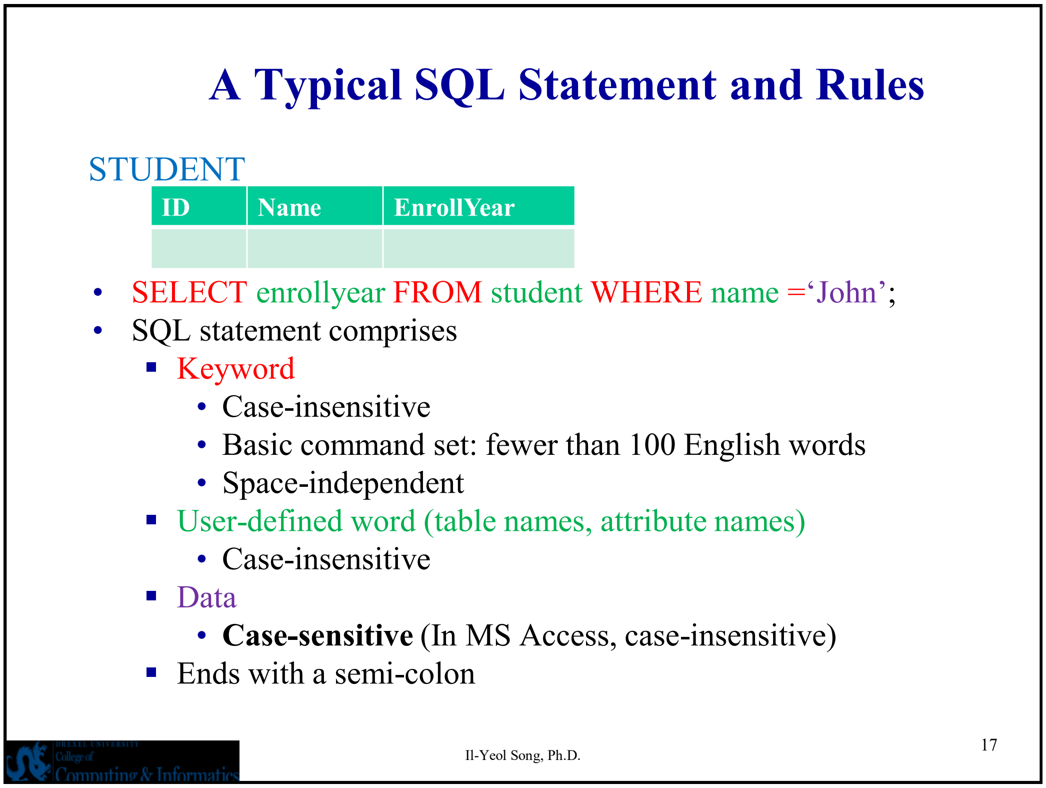
**Data Definition Language (DDL)**

- 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

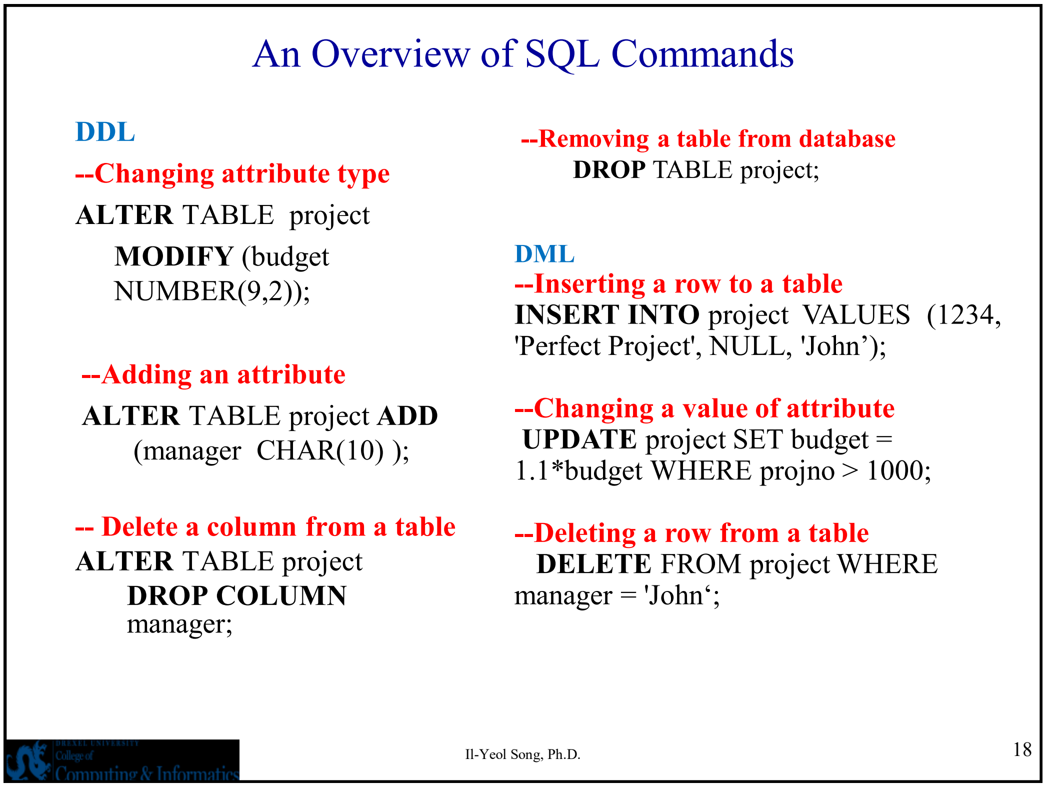


## A Typical SQL Statement and Rules

STUDENT

| ID | Name | EnrollYear |
|---|---|---|
|  |  |  |

- SELECT enrollyear FROM student WHERE name ='John';
- SQL statement comprises
  - Keyword
    - 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**

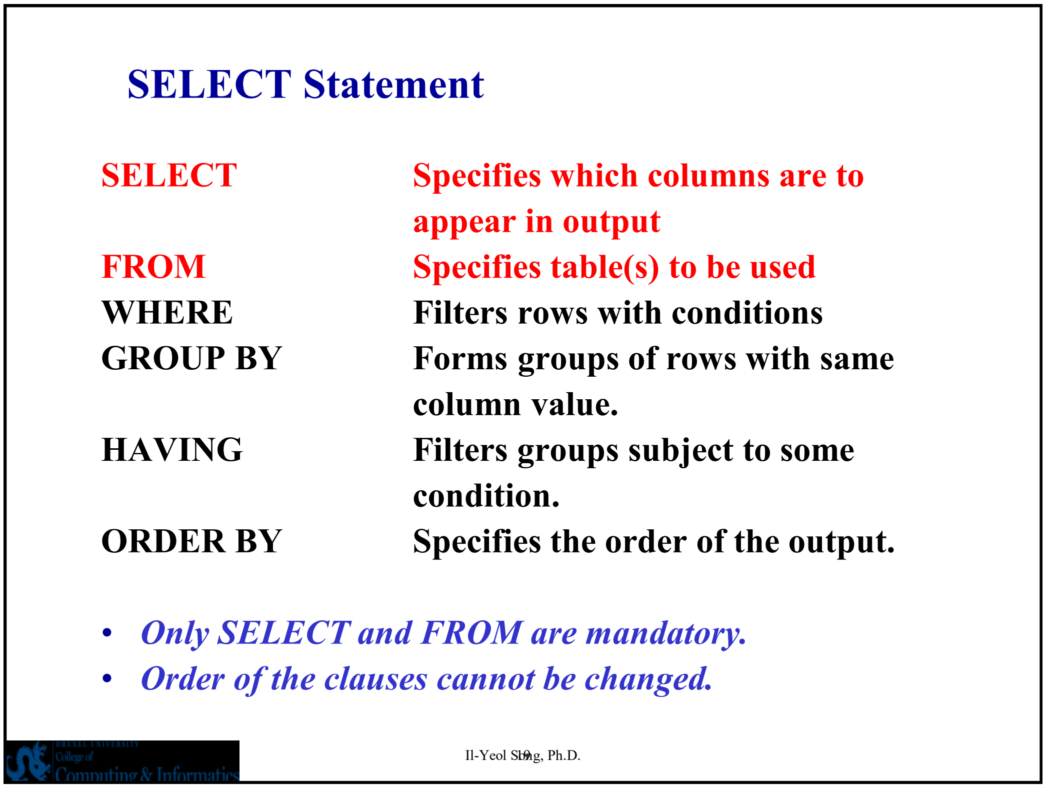
```
INSERT INTO project  VALUES  (1234,
'Perfect Project', NULL, 'John');
```

**--Changing a value of attribute**

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

**--Deleting a row from a table**

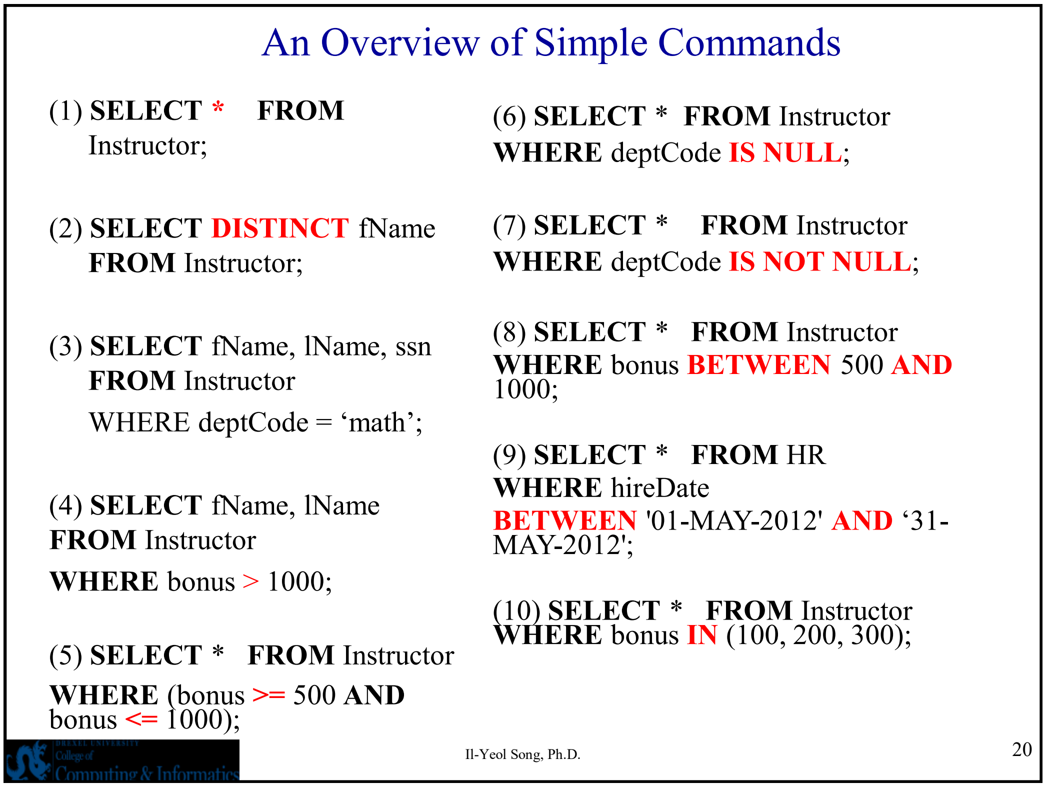
```
  DELETE FROM project WHERE
manager = 'John‘;
```



## 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 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);
```

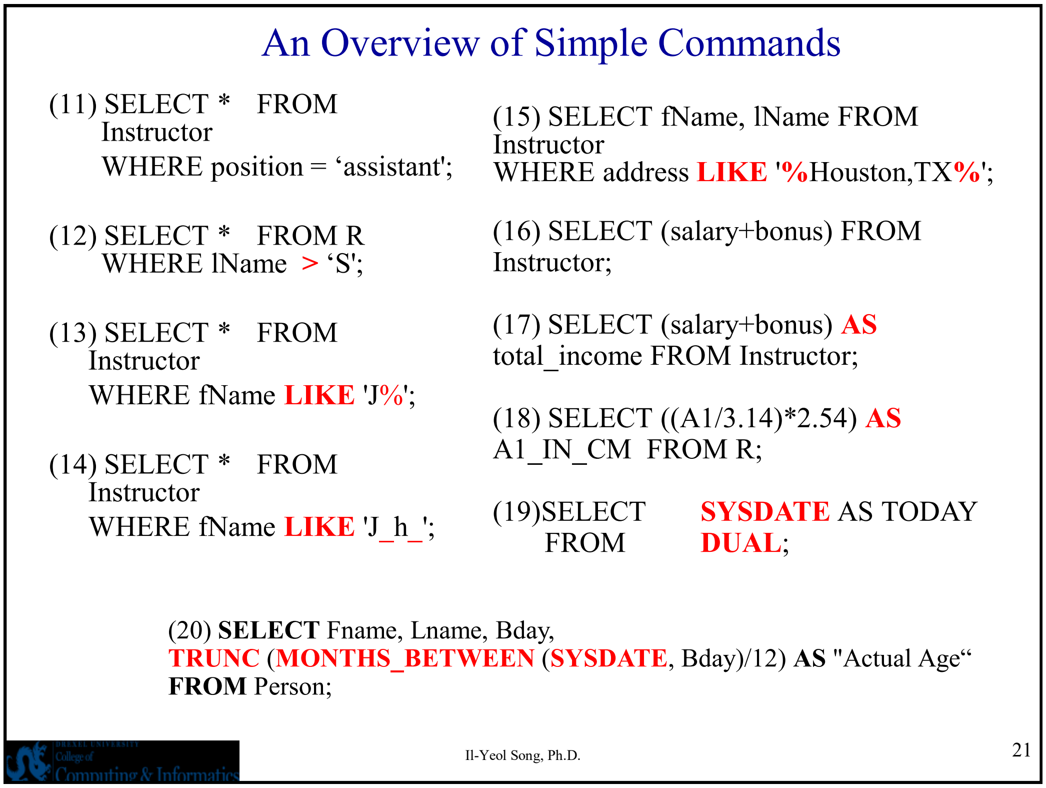
```
(6) SELECT *  FROM Instructor
WHERE deptCode IS NULL;
```

```
(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-2012' AND ‘31-
MAY-2012';
```

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

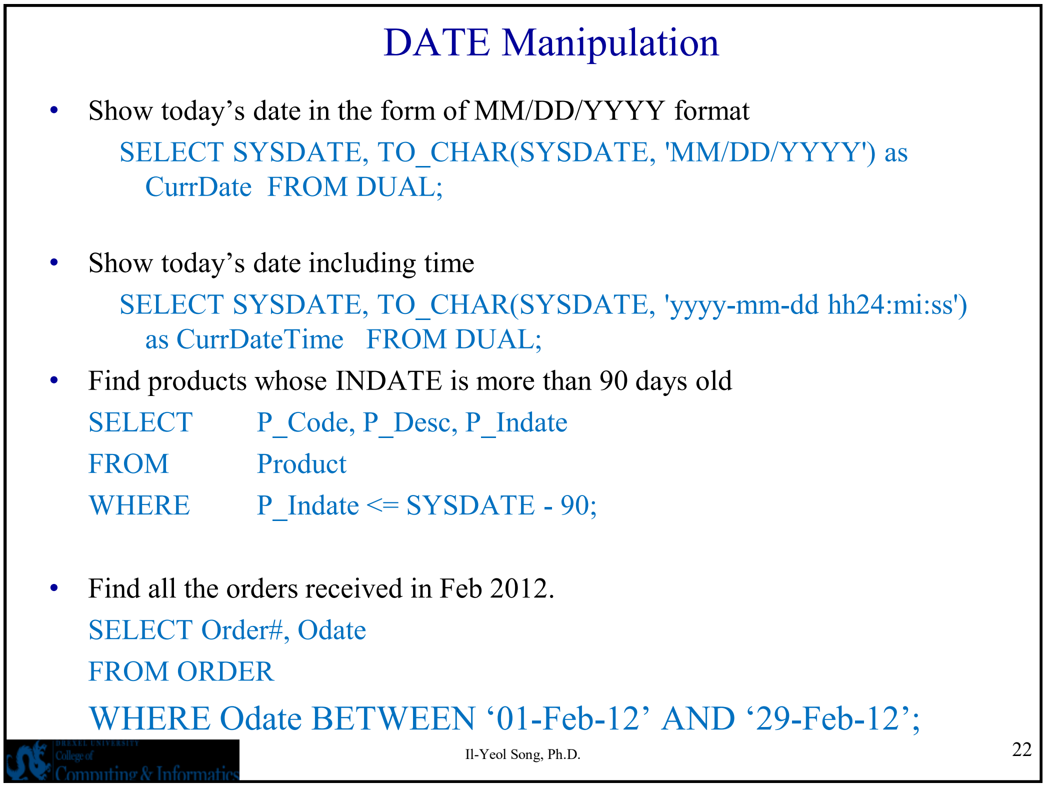


## An Overview of Simple Commands

```
(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;
```



## DATE Manipulation

- 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

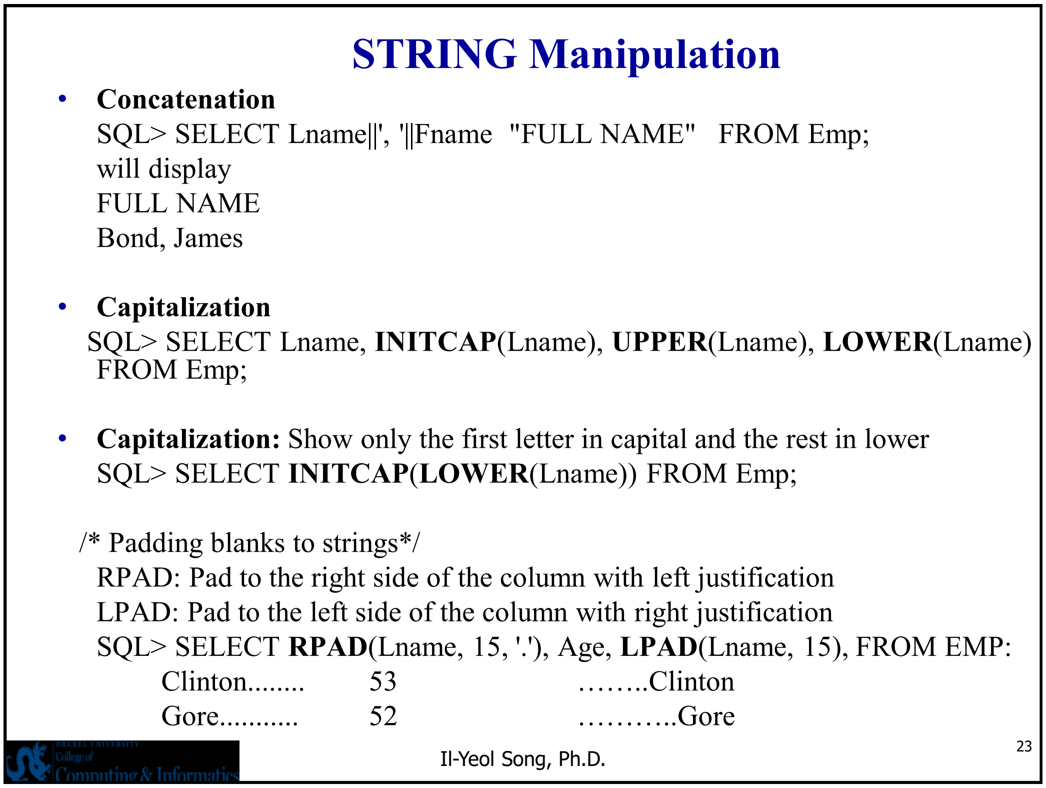
```
SELECT SYSDATE, TO_CHAR(SYSDATE, 'yyyy-mm-dd hh24:mi:ss')
  as CurrDateTime   FROM DUAL;
```

- 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 2012.

```
SELECT Order#, Odate
FROM ORDER
WHERE Odate BETWEEN ‘01-Feb-12’ AND ‘29-Feb-12’;
```



## STRING Manipulation

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

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

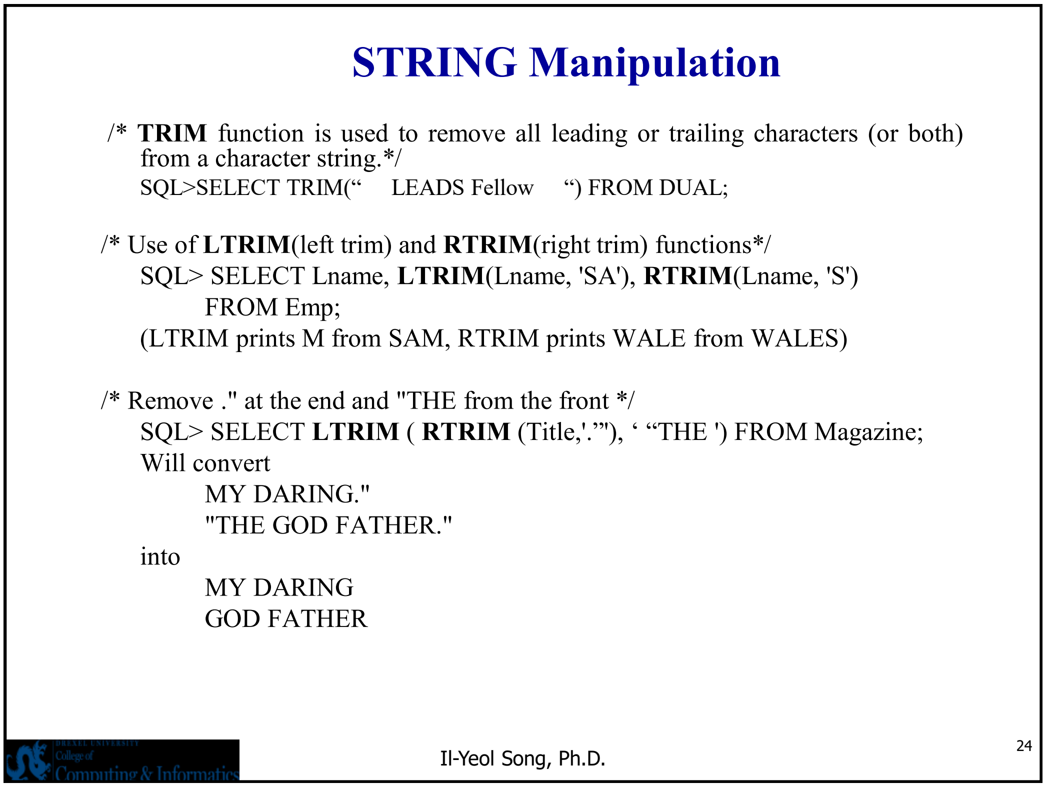
- **Capitalization:** Show only the first letter in capital and the rest in lower
  SQL> SELECT **INITCAP**(**LOWER**(Lname)) FROM Emp;

/* 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
```

Il-Yeol Song, Ph.D.

23



## STRING Manipulation

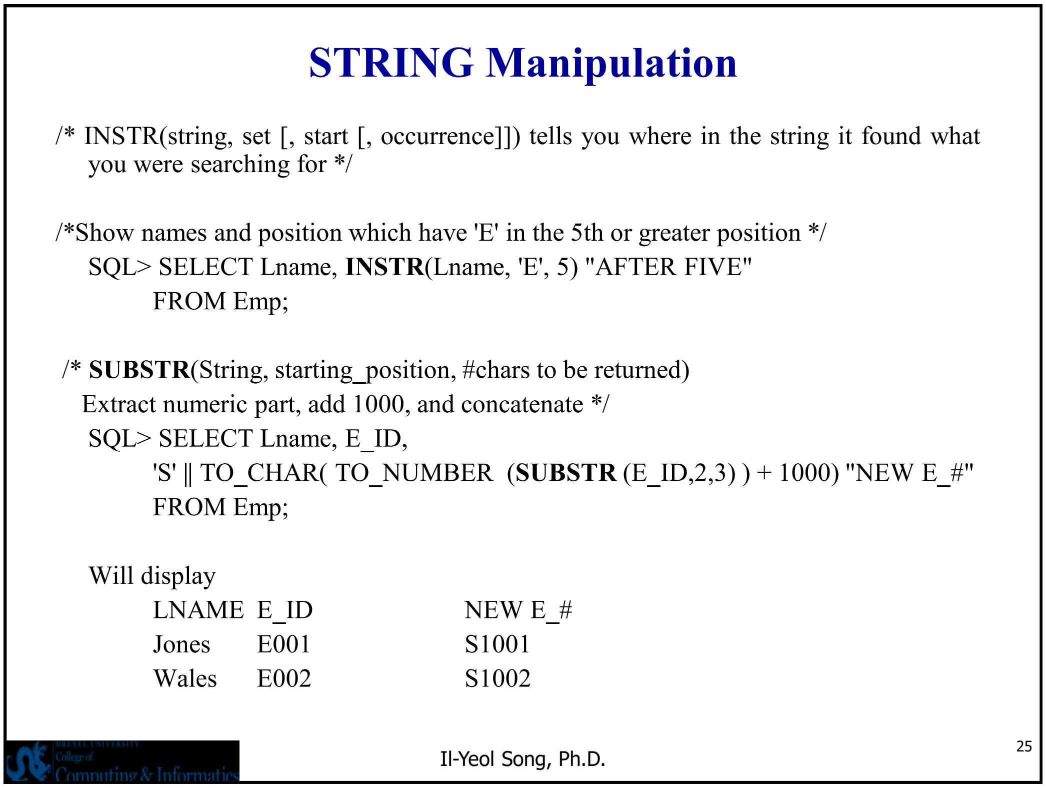
/* **TRIM** function is used to remove all leading or trailing characters (or both) from a character string.*/
SQL>SELECT TRIM(“     LEADS Fellow     “) FROM DUAL;

/* 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

Il-Yeol Song, Ph.D.

24



# STRING Manipulation

/* INSTR(string, set [, start [, occurrence]]) tells you where in the string it found what you were searching for */

/*Show names and position which have 'E' in the 5th or greater position */

```
SQL> SELECT Lname, INSTR(Lname, 'E', 5) "AFTER FIVE"
          FROM Emp;
```

/* **SUBSTR**(String, starting_position, #chars to be returned)
Extract numeric part, add 1000, and concatenate */

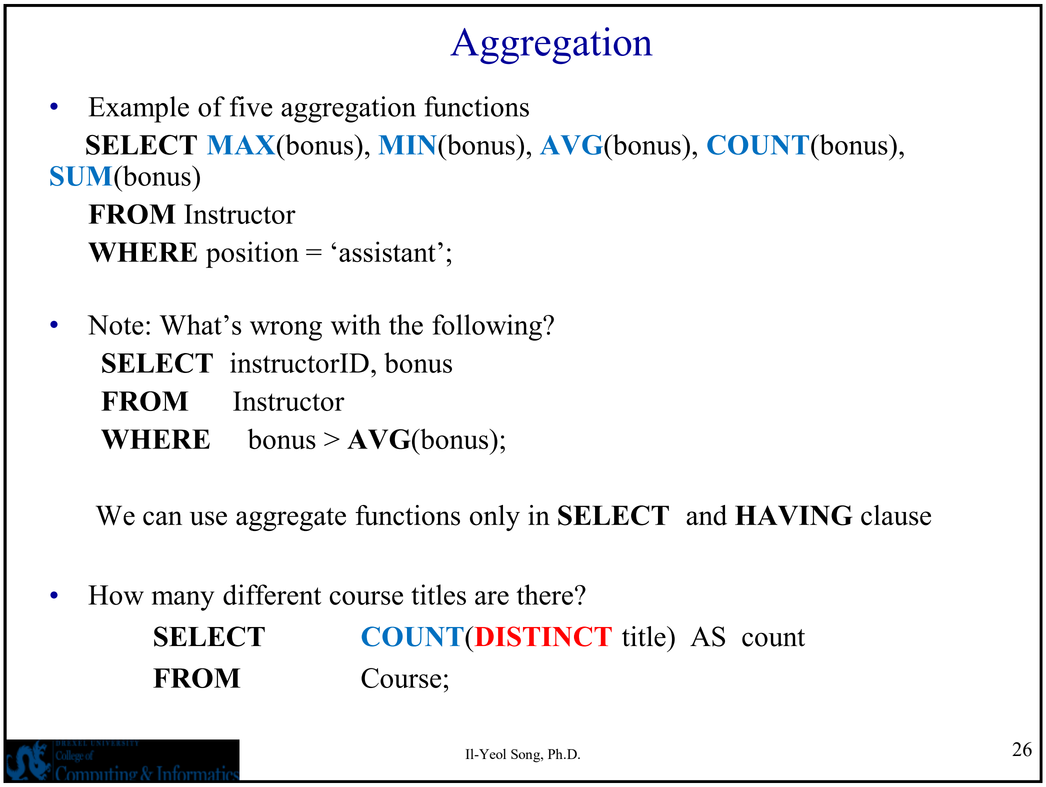
```
SQL> SELECT Lname, E_ID,
          'S' || TO_CHAR( TO_NUMBER  (SUBSTR (E_ID,2,3) ) + 1000) "NEW E_#"
          FROM Emp;
```

Will display

```
LNAME  E_ID            NEW E_#
Jones     E001            S1001
Wales     E002            S1002
```

Il-Yeol Song, Ph.D.

25



## Aggregation

- Example of five aggregation functions

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

- Note: What’s wrong with the following?

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

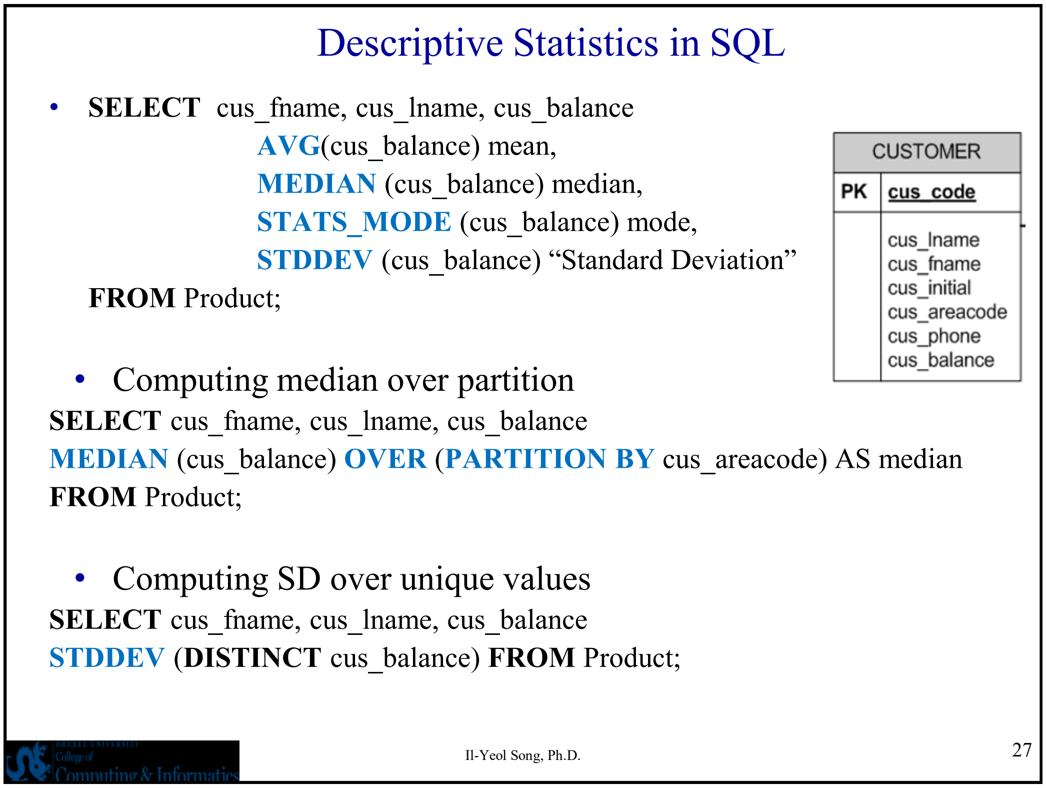
We can use aggregate functions only in **SELECT** and **HAVING** clause

- How many different course titles are there?

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

Il-Yeol Song, Ph.D.

26



## 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;

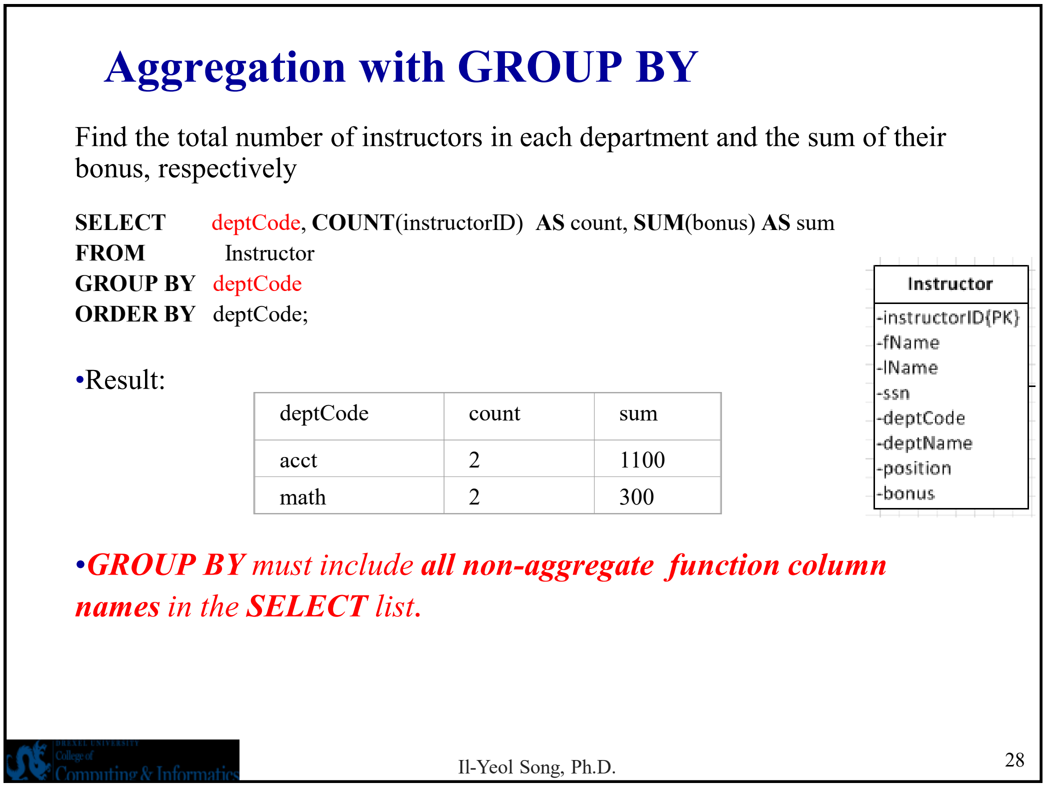
| CUSTOMER | |
|---|---|
| PK | <u>cus_code</u> |
| | cus_lname<br>cus_fname<br>cus_initial<br>cus_areacode<br>cus_phone<br>cus_balance |

- 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;
```

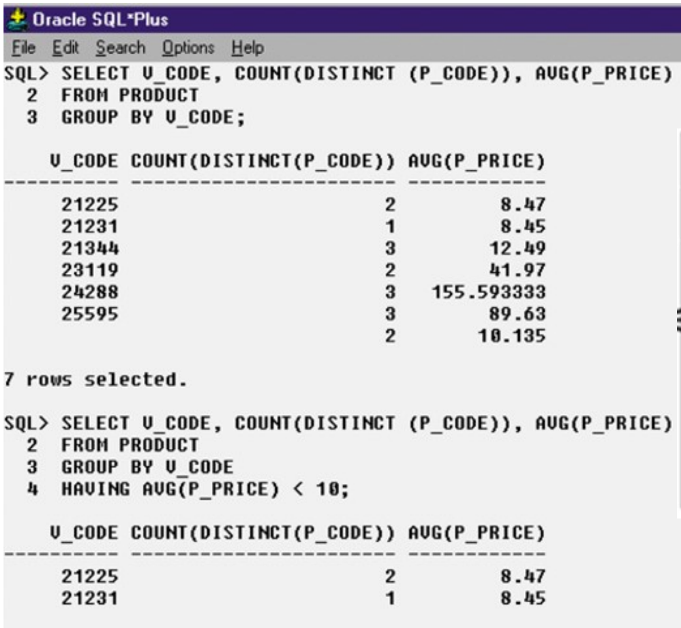
| Instructor |
|---|
| -instructorID{PK}<br>-fName<br>-lName<br>-ssn<br>-deptCode<br>-deptName<br>-position<br>-bonus |

•Result:

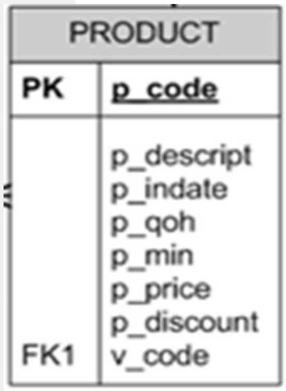
| deptCode | count | sum |
|---|---|---|
| acct | 2 | 1100 |
| math | 2 | 300 |

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

## Aggregation with GROUP BY and HAVING



```
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
```

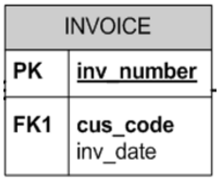


| PRODUCT | |
|---|---|
| PK | p_code |
| | p_descript<br>p_indate<br>p_qoh<br>p_min<br>p_price<br>p_discount |
| FK1 | v_code |

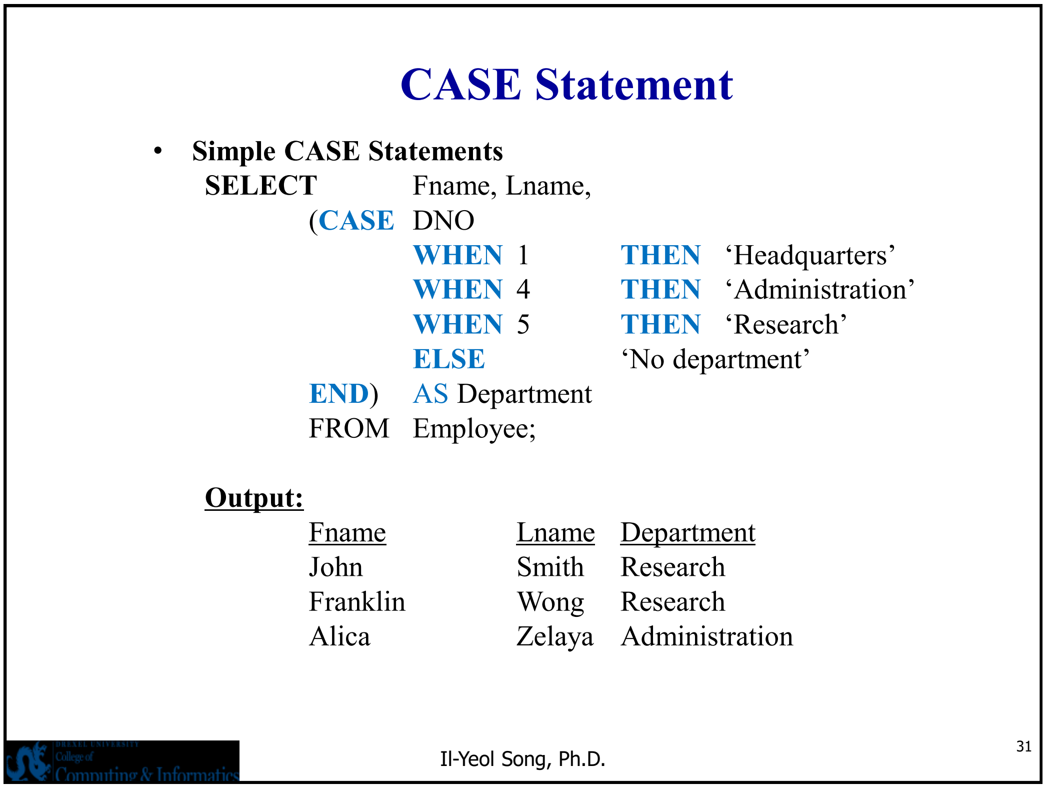
## Aggregation with GROUP BY and HAVING

- 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<br>inv_date |



## CASE Statement

- **Simple CASE Statements**

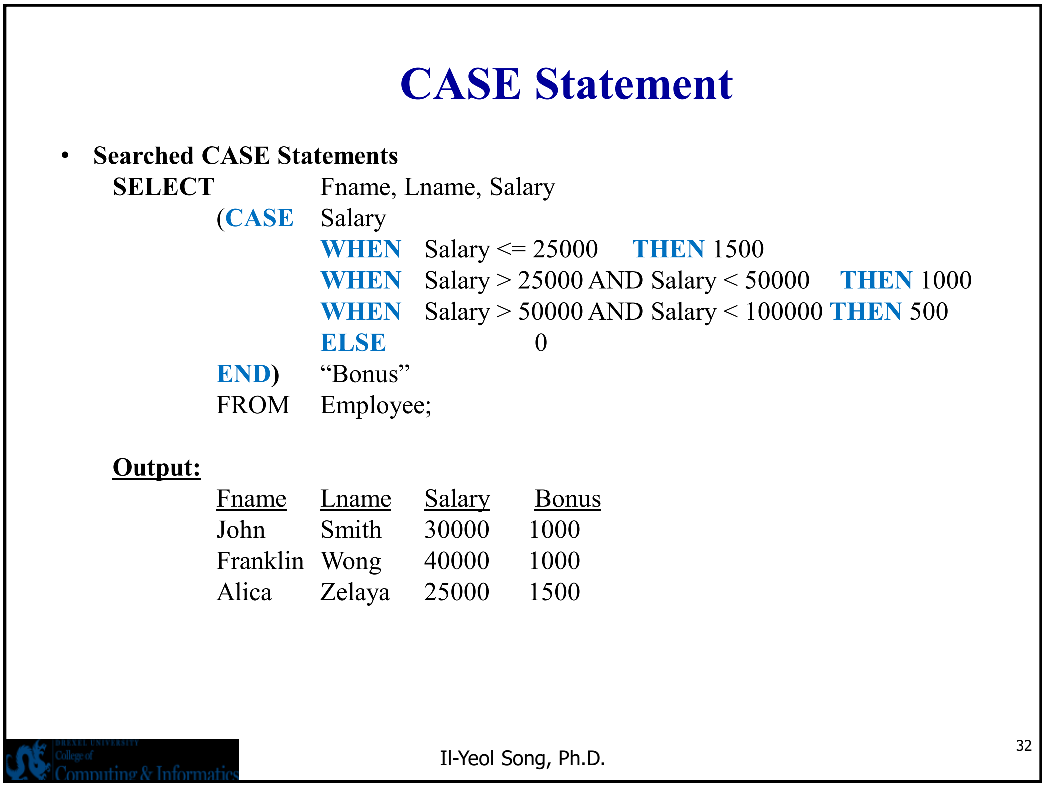
```
SELECT      Fname, Lname,
        (CASE  DNO
               WHEN 1      THEN  'Headquarters'
               WHEN 4      THEN  'Administration'
               WHEN 5      THEN  'Research'
               ELSE              'No department'
        END)   AS Department
        FROM   Employee;
```

**Output:**

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

Il-Yeol Song, Ph.D.

31



## 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
```

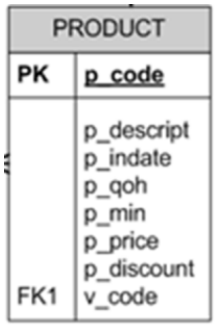
Il-Yeol Song, Ph.D.

32

## 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);
```



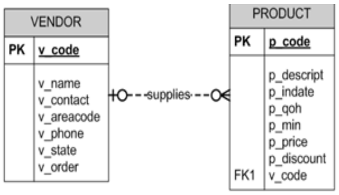
This query is in the form of a nested query.

## 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

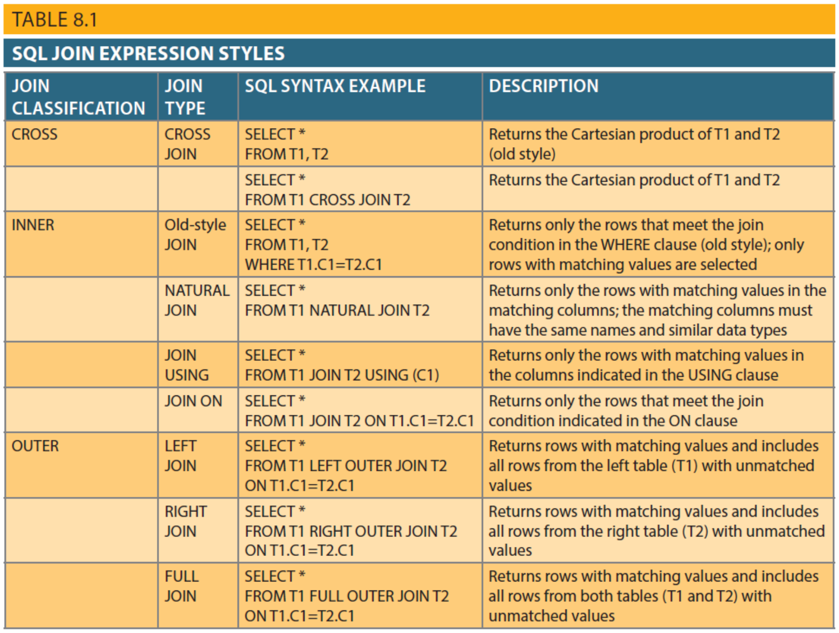


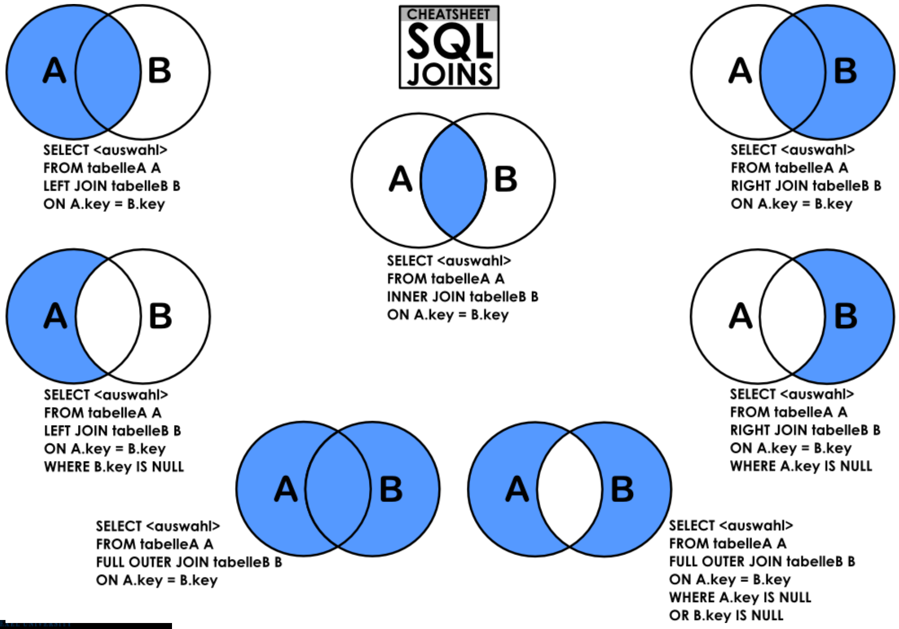
TABLE 8.1

SQL JOIN EXPRESSION STYLES

| JOIN CLASSIFICATION | JOIN TYPE | SQL SYNTAX EXAMPLE | DESCRIPTION |
|---|---|---|---|
| CROSS | CROSS JOIN | SELECT *<br>FROM T1, T2 | Returns the Cartesian product of T1 and T2 (old style) |
| | | SELECT *<br>FROM T1 CROSS JOIN T2 | Returns the Cartesian product of T1 and T2 |
| INNER | Old-style JOIN | SELECT *<br>FROM T1, T2<br>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 *<br>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 *<br>FROM T1 JOIN T2 USING (C1) | Returns only the rows with matching values in the columns indicated in the USING clause |
| | JOIN ON | SELECT *<br>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 *<br>FROM T1 LEFT OUTER JOIN T2<br>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 *<br>FROM T1 RIGHT OUTER JOIN T2<br>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 *<br>FROM T1 FULL OUTER JOIN T2<br>ON T1.C1=T2.C1 | Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values |

*(Coronel & Morris, 2016)*

# Visualizing JOIN Operations



```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```

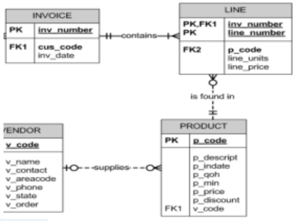
```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```

```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```

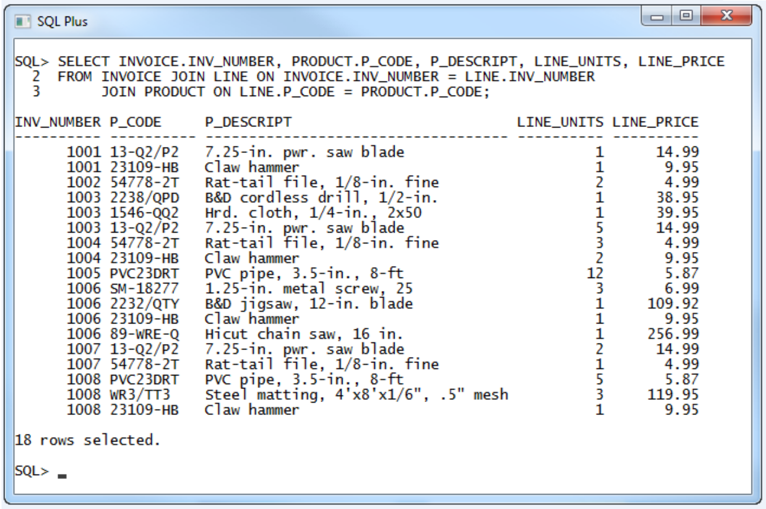
```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Il-Yeol Song, Ph.D.

# INNER JOIN



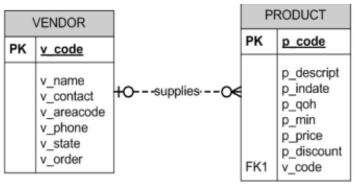
- For each invoice, find the product description, #units, and price of items in the invoice.



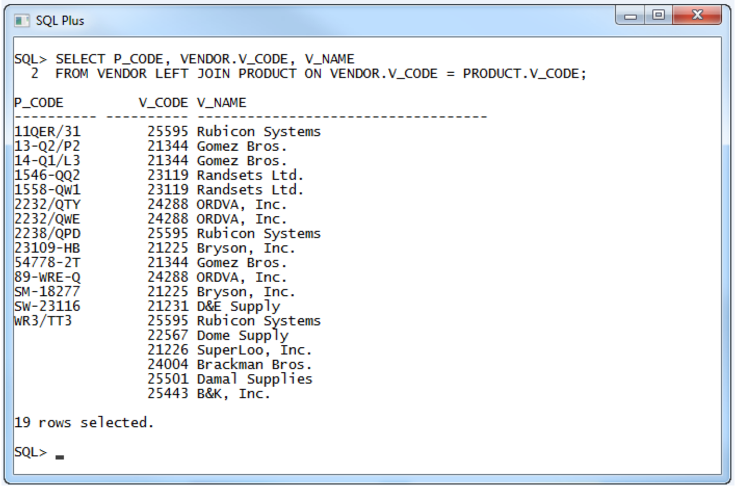
```
SQL> SELECT INVOICE.INV_NUMBER, PRODUCT.P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE
  2  FROM INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
  3       JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;

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> _
```

*(Coronel & Morris, 2016)*

# LEFT OUTER JOIN



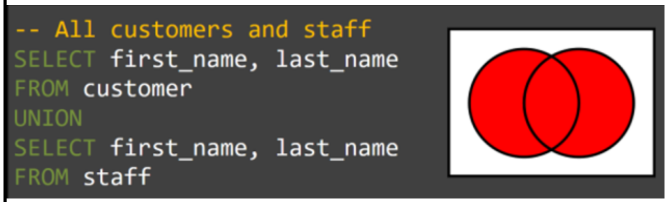
•For ALL the Vendors, find all the products they provide. Include all the vendors whether or not they provide any product.



```
SQL> 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> _
```

*(Coronel & Morris, 2016)*

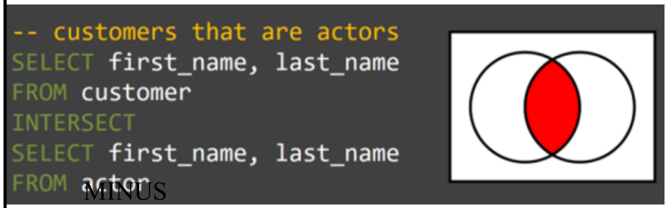
## SET Operations



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

UNION

R ∪ S

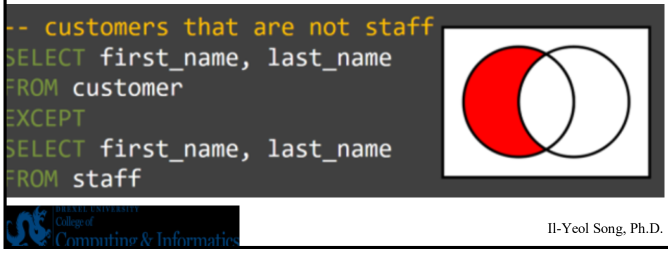


```
-- customers that are actors
SELECT first_name, last_name
FROM customer
INTERSECT
SELECT first_name, last_name
FROM actor
```

MINUS

INTERSECT

R ∩ S



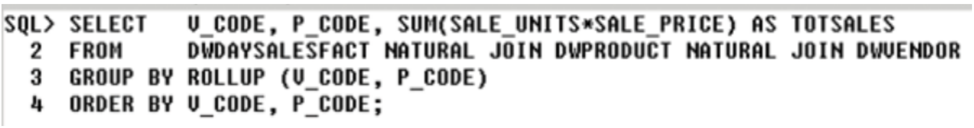
```
-- customers that are not staff
SELECT first_name, last_name
FROM customer
EXCEPT
SELECT first_name, last_name
FROM staff
```

MINUS

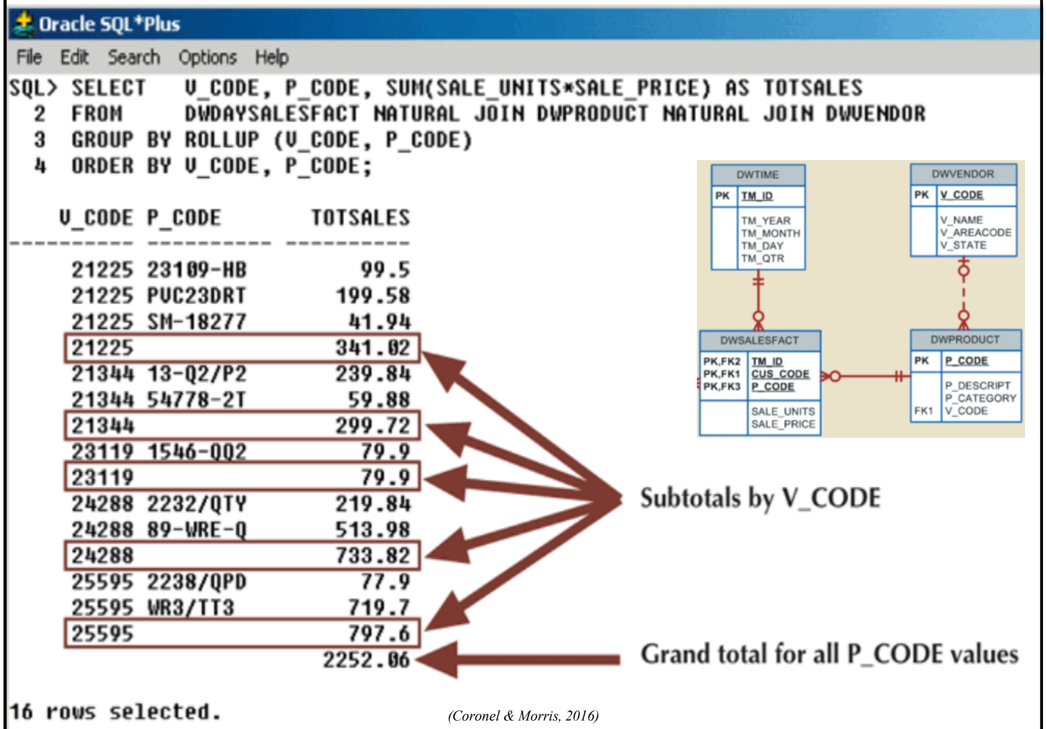
R − S

## SQL Analytic 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;
```



```
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.
```

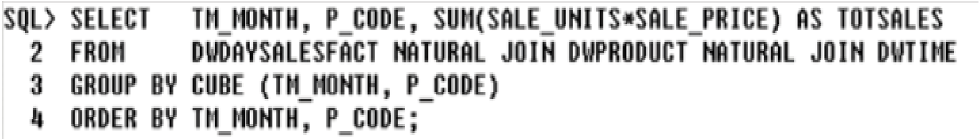
Subtotals by V_CODE

Grand total for all P_CODE values

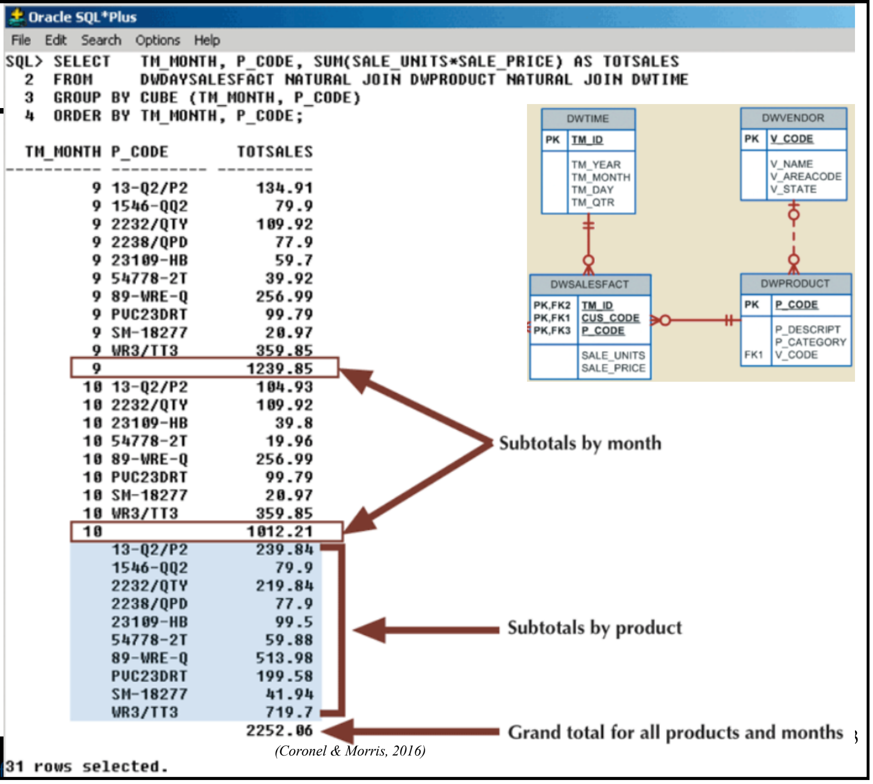
(Coronel & Morris, 2016)

## SQL Analytic 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 CUBE (TM_Month, P_CODE)
    - GROUP BY CUBE (TM_Month)
    - GROUP BY CUBE (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;
```



```
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 PVC23DRT        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 PVC23DRT        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
           PVC23DRT       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

(Coronel & Morris, 2016)

# RANK and DENSE RANK Functions

- RANK() leave ranking gaps when there 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;
```

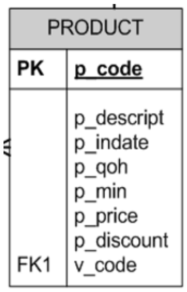
| PRODUCT | |
|---|---|
| PK | p_code |
| | p_descript<br>p_indate<br>p_qoh<br>p_min<br>p_price<br>p_discount |
| FK1 | v_code |

- 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

- **Before Oracle 12c:**

```
SELECT  p_code, p_descript,
            (SELECT p_price,
                        RANK() OVER
                            (ORDER BY p_price NULLS LAST) AS Rank
             FROM product)
FROM     Product
WHERE  Rank <= 10
ORDER BY p_price;
```

- **In Oracle 12c:**

```
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
```

# Question?