Big Data Landscape: SQL Systems

by Asst. Prof. Kulsawasd Jitkajornwanich, Ph.D.

KMITL Digital Analytics and Intelligence Center

[Adapted from slides and/or other materials by Bengfort & Kim "Data Analytics with Hadoop", White "Hadoop: The Definitive Guide", Schutt & O'Neil "Doing Data Science", Elmasri & Navathe "Fundamentals of Database Systems", Cielen et al. "Introducing Data Science", and Provost & Fawcett "Data Science for Business"]

Asst.Prof.Dr.

Kulsawasd Jitkajornwanich

- Received Master's and PhD degrees in Computer Science from The University of Texas at Arlington, USA
  - **Dissertation:** "Analysis and Modeling Techniques for Geo-Spatial and Spatio-Temporal Datasets"
- Conducting research/projects and academic services in the areas of database, NOSQL systems, big data analytics, and custom location-based applications

Course outline

- Introduction to Related Terminology
- SQL Data Storage Platform
- Lab exercises on SQL concepts
- Conclusion
- Review questions & Q/A

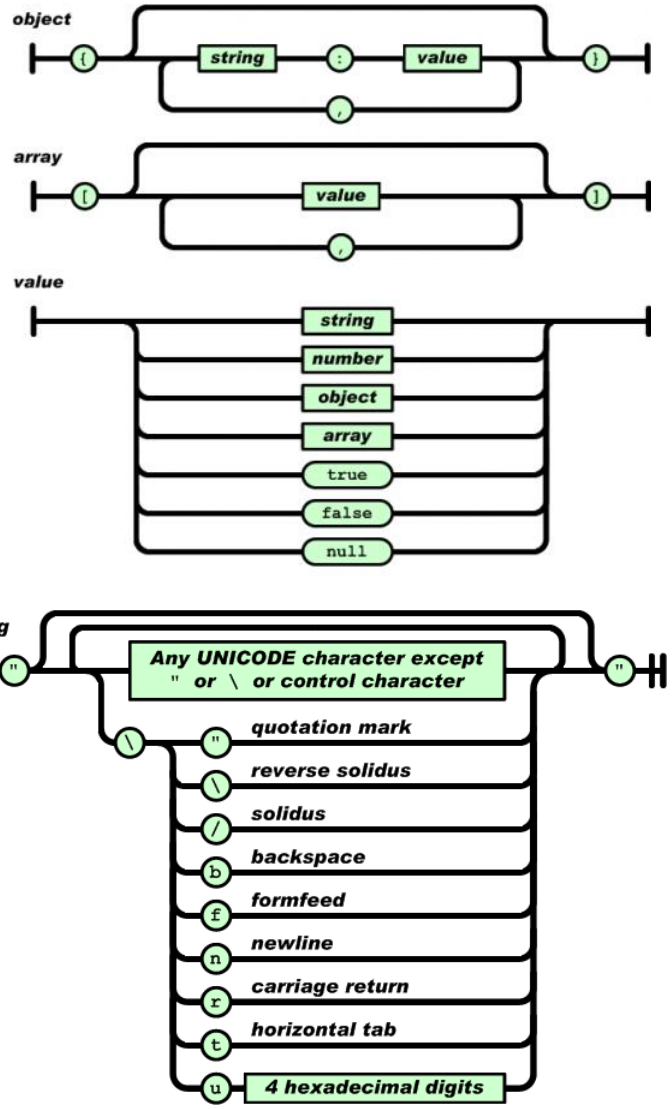
Types of (Big) Data

	Indicator ID	Dimension List	Timeframe	Numeric Value	Missing Value Flag	Confidence Inte
2	214390830	Total (Age-adjusted)	2008	74.6%		73.8%
3	214390833	Aged 18-44 years	2008	59.4%		58.0%
4	214390831	Aged 18-24 years	2008	37.4%		34.6%
5	214390832	Aged 25-44 years	2008	66.9%		65.5%
6	214390836	Aged 45-64 years	2008	88.6%		87.7%
7	214390834	Aged 45-54 years	2008	86.3%		85.1%
8	214390835	Aged 55-64 years	2008	91.5%		90.4%
9	214390840	Aged 65 years and over	2008	94.6%		93.8%
10	214390837	Aged 65-74 years	2008	93.6%		92.4%
11	214390838	Aged 75-84 years	2008	95.6%		94.4%
12	214390839	Aged 85 years and over	2008	96.0%		94.0%
13	214390841	Male (Age-adjusted)	2008	72.2%		71.1%
14	214390842	Female (Age-adjusted)	2008	76.8%		75.9%
15	214390843	White only (Age-adjusted)	2008	73.8%		72.9%
16	214390844	Black or African American only (Age-adjusted)	2008	77.0%		75.0%
17	214390845	American Indian or Alaska Native only (Age-adjusted)	2008	66.5%		57.1%
18	214390846	Asian only (Age-adjusted)	2008	80.5%		77.7%
19	214390847	Native Hawaiian or Other Pacific Islander only (Age-adjusted)	2008	DSU		
20	214390848	2 or more races (Age-adjusted)	2008	75.6%		69.6%

Figure 1.1 An Excel table is an example of structured data.

```
CSIPERF:TXCOMMIT;313236
2014-11-28 11:36:13, Info                CSI    00000153 Creating NT transaction (seq
69), objectname [6]"(null)"
2014-11-28 11:36:13, Info                CSI    00000154 Created NT transaction (seq 69)
result 0x00000000, handle @0x4e54
2014-11-28 11:36:13, Info                CSI    00000155@2014/11/28:10:36:13.471
Beginning NT transaction commit...
2014-11-28 11:36:13, Info                CSI    00000156@2014/11/28:10:36:13.705 CSI perf
trace:
CSIPERF:TXCOMMIT;273983
2014-11-28 11:36:13, Info                CSI    00000157 Creating NT transaction (seq
70), objectname [6]"(null)"
2014-11-28 11:36:13, Info                CSI    00000158 Created NT transaction (seq 70)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:13, Info                CSI    00000159@2014/11/28:10:36:13.764
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                CSI    0000015a@2014/11/28:10:36:14.094 CSI perf
trace:
CSIPERF:TXCOMMIT;386259
2014-11-28 11:36:14, Info                CSI    0000015b Creating NT transaction (seq
71), objectname [6]"(null)"
2014-11-28 11:36:14, Info                CSI    0000015c Created NT transaction (seq 71)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:14, Info                CSI    0000015d@2014/11/28:10:36:14.106
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                CSI    0000015e@2014/11/28:10:36:14.428 CSI perf
trace:
CSIPERF:TXCOMMIT;375581
```

Figure 1.3 Example of machine-generated data



SQL System Basics

- SQL stands for **S**tructured **Q**uery **L**anguage (Originally called SEQUEL: **S**tructured **E**nglish **Qu**ery **L**anguage)
  - Developed at IBM Research for experimental **Relational DBMS** called *System-R* in the 1970s
- SQL is a standard, comprehensive language, based on the relational model
- Considered one of the major reasons for the commercial success of relational databases
- **2 main capabilities of SQL discussed today:**
  - 1) CREATING schemas and specifying data types and constraints
  - 2) Specifying database retrievals (QUERIES)

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Continued next page...

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

1) Create statement

- **CREATE** statement
  - Main SQL command for data definition
- CREATE (DDL) statement can be used to:
  - Create a named database schema
  - Create individual database tables and specify data types for the table attributes, as well as *key*, *referential integrity*, and *NOT NULL* constraints
  - Create named constraints
- Other commands can modify the tables and constraints
  - DROP and ALTER statements
    - CASCADE | RESTRICT (tables must be empty) options are also applied
    - For add'l commands (e.g., ADD COLUMN, DROP COLUMN, ALTER COLUMN, DROP CONSTRAINT), see Table 7.2 last two slides

CREATE TABLE

- In its *simplest form*, specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n), DATE, and other data types)
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT (
 DNAME           VARCHAR(15)  NOT NULL,
 DNUMBER         INT          NOT NULL,
 MGRSSN          CHAR(9)      NOT NULL,
 MGRSTARTDATE    DATE    );
```

CREATE TABLE (cont.)

- CREATE TABLE can also specify the primary key, UNIQUE keys, and referential integrity constraints (foreign keys)
- Via the PRIMARY KEY, UNIQUE, and FOREIGN KEY phrases

```
CREATE TABLE DEPARTMENT (
DNAME          VARCHAR(15)  NOT NULL,
DNUMBER        INT          NOT NULL,
MGRSSN         CHAR(9)      NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY(MGRSSN) REFERENCES EMPLOYEE(SSN));
```

Example: The COMPANY Database

- Figure in the next slide shows the COMPANY database *schema diagram*
- Referential integrity constraints shown as directed edges (-->) from *foreign key* to *referenced relation (or parent table)*
- Primary key attributes of each table *underlined*

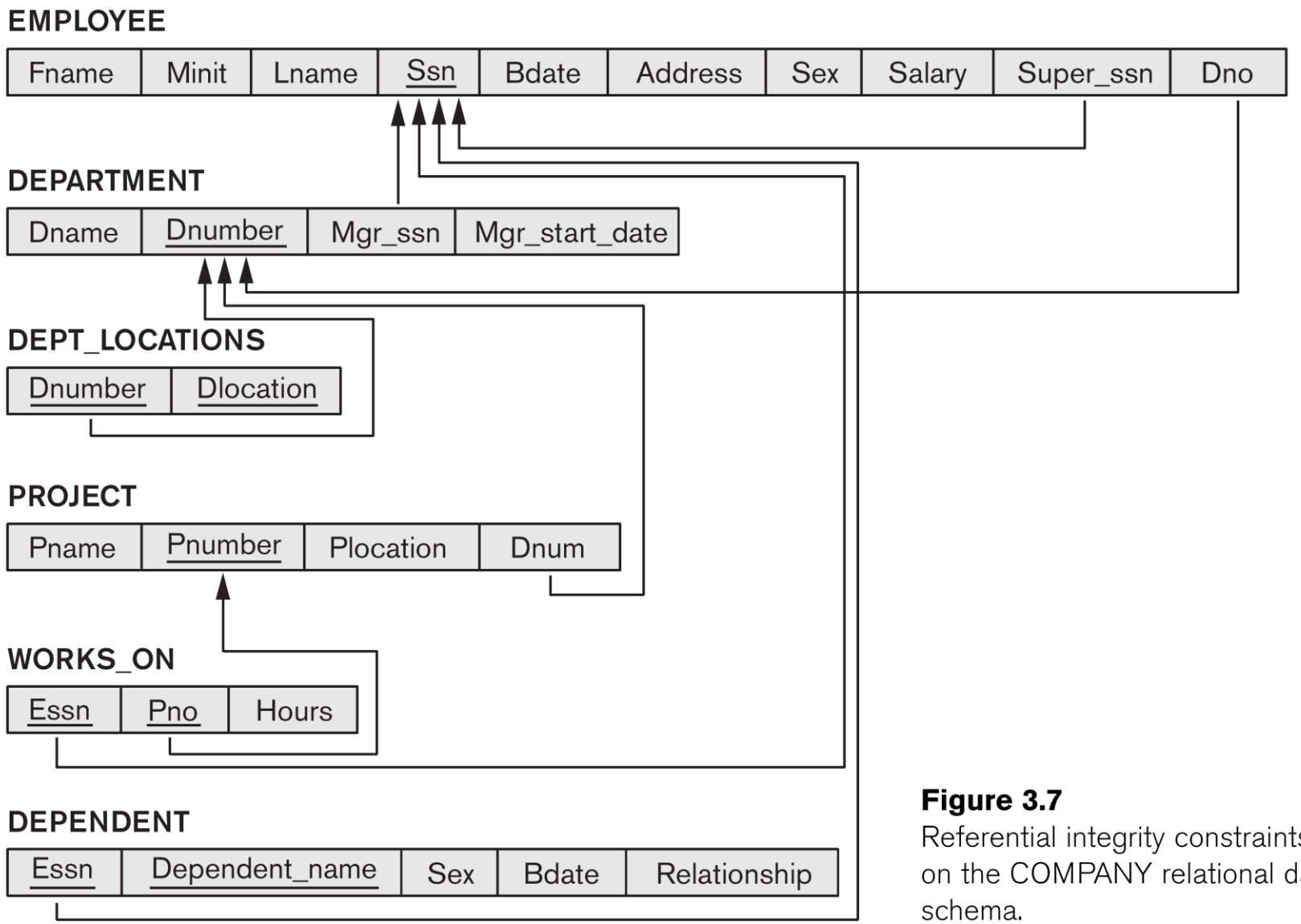


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)          NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)          NOT NULL,
          Ssn            CHAR(9)              NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE DEPARTMENT
        ( Dname          VARCHAR(15)          NOT NULL,
          Dnumber        INT                  NOT NULL,
          Mgr_ssn        CHAR(9)              NOT NULL,
          Mgr_start_date DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber        INT                  NOT NULL,
          Dlocation      VARCHAR(15)          NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

Continued next page…

```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Basic SQL Data Types

- Basic numeric data types:
  - Integers: INTEGER (or INT), SMALLINT
  - Real (floating point): FLOAT (or REAL), DOUBLE PRECISION
  - Formatted: DECIMAL(i,j) (or DEC (i,j) or NUMERIC(i,j)) specify i total decimal digits, j after decimal point
    - i called *precision*, j called *scale*
- Basic character string data types:
  - Fixed-length: CHAR(n) or CHARACTER(n)
  - Variable-length: VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n)

SQL Data Types (cont)

- DATE data type:
  - Standard DATE formatted as yyyy-mm-dd
  - For example, DATE '2010-01-22'
  - Older formats still used in some systems, such as 'JAN-22-2010'
  - Values are ordered, with later dates larger than earlier ones
- TIME data type:
  - Standard TIME formatted as hh:mm:ss
  - E.g., TIME '15:20:22' represents 3.20pm and 22 seconds
  - TIME(i) includes an additional i-1 decimal digits for fractions of a second
  - E.g., TIME(5) value could be '15:20:22.1234'
- TIMESTAMP data type:
  - A DATE combined with a TIME(i), where the default i=7
  - For example, TIMESTAMP '2010-01-22 15:20:22.123456'
  - A different i>7 can be specified if needed

2) Read statement (Queries)

- Simplest form of the SQL SELECT statement is called a *mapping* or a SELECT-FROM-WHERE *block*
- Our examples use the COMPANY database schema

- **SELECT** <attribute list>
  FROM <table list>
  WHERE <condition>

  - <attribute list> is a list of attribute names whose values are to be retrieved by the query
  - <table list> is a list of the table (relation) names required to process the query
  - <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Simple SQL Queries (cont.)

- Query text ends with a semi-colon
- First example of a simple query on one table (relation)
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'
- Use the EMPLOYEE table only

```
Q0:   SELECT    BDATE, ADDRESS
      FROM          EMPLOYEE
      WHERE         FNAME='John' AND MINIT='B'
                    AND LNAME='Smith' ;
```

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Continued next page...

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Simple SQL Queries (cont.)

- Example of a query that uses two tables
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT    FNAME, LNAME, ADDRESS
     FROM      EMPLOYEE, DEPARTMENT
     WHERE     DNAME='Research' AND DNUMBER=DNO ;
```

  - (DNAME='Research') is called a *selection condition*
  - (DNUMBER=DNO) is called a *join condition* (it *joins* two tuples from EMPLOYEE and DEPARTMENT tables whenever EMPLOYEE.DNO=DEPARTMENT.DNUMBER)

Simple SQL Queries (cont.)

- A *selection condition* refers to attributes from a single table, and selects (chooses) those records that satisfy the condition
- A *join condition generally* refers to attributes from two tables, and joins (or combines) pairs of records that satisfy the condition
- In the previous query:
  - (DNAME='Research') chooses the DEPARTMENT record
  - (DNUMBER=DNO) joins the record with each EMPLOYEE record that works for that department

Aliasing Table Names Using Tuple Variables

- An *alias* (or *tuple variable*) can be used instead of the table name when prefixing attribute names
- Example:

Query Q1 can be rewritten as follows using the aliases D for DEPARTMENT and E for EMPLOYEE:

```
SELECT    E.FNAME, E.LNAME, E. ADDRESS
FROM      EMPLOYEE AS E, DEPARTMENT AS D
WHERE     D.DNAME='Research' AND D.DNUMBER=E.DNO ;
```

Missing WHERE-clause

- The *WHERE-clause* is **optional** in an SQL query
- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE
- Example: Retrieve the SSN values for all employees.

```
Q9:  SELECT   SSN
     FROM     EMPLOYEE ;
```

- If more than one relation is specified in the FROM-clause *and* there is no WHERE-clause (hence no join conditions), then *all possible combinations* of tuples are joined together (known as *CARTESIAN PRODUCT* of the relations)

Retrieving all the Attributes Using Asterisk (*)

- To retrieve all the attribute values of the selected tuples, a * (asterisk) is used, which stands for *all the attributes*

  Examples:

```
Q1C:  SELECT  *
      FROM    EMPLOYEE
      WHERE   DNO=5 ;

Q1D:  SELECT  *
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNAME='Research' AND
              DNO=DNUMBER ;
```

Eliminating Duplicates Using DISTINCT

- As mentioned earlier, SQL does not treat a relation as a set but a multiset (or bag); duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- Example: Result of Q11 may have duplicate SALARY values; result of Q11A does not have any duplicate values

```
Q11:    SELECT   SALARY
        FROM     EMPLOYEE

Q11A:   SELECT   DISTINCT SALARY
        FROM     EMPLOYEE
```

Substring Comparison Conditions

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '*****' (or '**%**' in some implementations) replaces an arbitrary number of characters, and '**_**' replaces a single arbitrary character
- Conditions can be used in WHERE-clause

Substring Comparison Conditions (cont.)

- Example: Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX‘ in it.

```
Q25:    SELECT  FNAME, LNAME
        FROM    EMPLOYEE
        WHERE   ADDRESS LIKE '*Houston,TX*' ;
```

Substring Comparison Conditions (cont.)

- Example: Query 26: Retrieve all employees who were born during the 1950s.
  - Here, '5' must be the 3rd character of the string (according to the standard format for DATE yyyy-mm-dd), so the BDATE value is
    '- - 5 - - - - - - -' with each underscore as a place holder for a single arbitrary character.

```
Q26:    SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     BDATE LIKE '- - 5 - - - - - - -'
```

- The LIKE operator allows users to get around the fact that each value is considered atomic and indivisible
  - Hence, in SQL, character string attribute values are not atomic

Applying Arithmetic in SQL Queries

- The standard arithmetic operators **'+', '–'. '*',** and **'/'** (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric attributes and values in an SQL query
- Example: Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27:    SELECT   FNAME, LNAME, 1.1*SALARY
        FROM     EMPLOYEE, WORKSON, PROJECT
        WHERE    SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Ordering a query result using the ORDER BY clause

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Example: Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28:    SELECT      DNAME, LNAME, FNAME, PNAME
        FROM        DEPARTMENT, EMPLOYEE, WORKSON, PROJECT
        WHERE       DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
        ORDER BY    DNAME, LNAME  DESC;
```

ORDER BY (cont.)

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default
- Without ORDER BY, the rows in a query result appear in some system-determined (random) order

Explicit (Literal) Sets in SQL

- An **explicit (enumerated) set of values** is enclosed in parentheses
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13: SELECT   DISTINCT ESSN
     FROM     WORKSON
     WHERE    PNO IN  (1, 2, 3) ;
```

Aggregate Functions

- Include **COUNT, SUM, MAX, MIN, and AVG**
- These can summarize information from multiple tuples into a single tuple
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15:    SELECT   MAX(SALARY) AS HIGHSAL,
                 MIN(SALARY) AS LOWSAL,
                 AVG(SALARY) AS MEANSAL
        FROM     EMPLOYEE ;
```

Aggregate Functions (cont.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16:    SELECT  MAX(E.SALARY), MIN(E.SALARY),  AVG(E.SALARY)
        FROM    EMPLOYEE E, DEPARTMENT D
        WHERE   E.DNO=D.DNUMBER   AND   D.DNAME='Research' ;
```

Aggregate Functions (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18). (Note: COUNT(*) counts the number of selected records)

```
Q17:    SELECT   COUNT (*)
        FROM     EMPLOYEE ;

Q18:    SELECT   COUNT (*)
        FROM     EMPLOYEE AS E, DEPARTMENT AS D
        WHERE    E.DNO=D.DNUMBER AND
                 D.DNAME='Research' ;
```

Grouping (Partitioning Records into Subgroups)

- In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation
- Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)* — for example, *employees who work in the same department* (have the same DNO)
- The aggregate functions are applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

Grouping (cont.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20:    SELECT      DNO, COUNT (*), AVG (SALARY)
        FROM        EMPLOYEE
        GROUP BY    DNO ;
```

  - In Q20, the EMPLOYEE tuples are divided into groups–
    - Each group has same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately (see next slide)
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples

Figure 5.1
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Continued next page...

Grouping (cont.)

- A join condition can be used with grouping
- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:    SELECT      P.PNUMBER, P.PNAME, COUNT (*)
        FROM        PROJECT AS P, WORKSON AS W
        WHERE       P.PNUMBER=W.PNO
        GROUP BY    P.PNUMBER, P.PNAME ;
```

  - In this case, the grouping and aggregate functions are applied *after* the joining of the two relations

The HAVING-Clause

- Sometimes we want to retrieve the values of these aggregate functions for only those *groups that satisfy certain conditions*
- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)
- Query 22: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project (Figure 5.1(b) — next two slides).

```
Q22:    SELECT      PNUMBER, PNAME, COUNT(*)
        FROM        PROJECT, WORKSON
        WHERE       PNUMBER=PNO
        GROUP BY    PNUMBER, PNAME
        HAVING      COUNT(*) > 2 ;
```

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

Summary of SQL Commands

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>] ;

Summary of SQL Commands (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries, as well as joined tables
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the query result
  - Conceptually, a query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause and ORDER BY

IN-CLASS EXERCISES

Now that you have learned the SQL system basics. Write SQL(s) to respond to the questions below on the sample database given in class.

a) Retrieve the names of all employees in department 5 who work more than 10 hours per week on the ProductX project.

b) List top 2 employee (ssn) who work the most hours across the projects

c) Find the names of all employees who are directly supervised by ‘Franklin Wong’.

d) Who is the manager of Research Department?

e) List the names of all employees who have salary more than their supervisor.

15–MIN BREAK

WE WILL BE BACK SOON.

Any Questions?