# COMP810

## LAB Week 1 Data Warehousing

Semester 2 2024

## Basic SQL Commands

# Tables in SQL

Table name

Attribute names

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Tuples or rows

# ‘Schema’ of an SQL Table & Key attributes

- The *schema* of a table is the table name and its attributes – Notation:

Product(PName, Price, Category, Manfacturer)

- A *key* is an attribute whose values are unique; we underline a key – Notation:

Product(<u>PName</u>, Price, Category, Manfacturer)

# DDL

The SQL data-definition language (DDL) allows the specification of information about relations (tables), including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints

# DOMAIN Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric**(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.

# CREATE TABLE command

- An SQL relation is defined using the **create table** command:

  **create table** *r* ($A_1$ $D_1$, $A_2$ $D_2$, ..., $A_n$ $D_n$,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the **name** of the relation (character string)
  - each $A_i$ is an **attribute** name in the schema of relation *r*
  - $D_i$ is the data type of values in the domain of attribute $A_i$

- Example:

```
create table instructor (
        ID          char(5),
        name        varchar(20),
        dept_name   varchar(20),
        salary      numeric(8,2))
```

# Integrity Constraints: Primary Key vs Foreign Key

- An SQL relation is defined using the **create table** command:

```
create table instructor (
        ID          char(5),
        name        varchar(20),
        dept_name   varchar(20),
        salary      numeric(8,2),
        primary key ( ZZZZ, ZZZZ, ZZZ ),
        foreign key (ZZZZ) references r)
```

- A primary key uniquely identifies a row in a table. Cannot have a NULL value
- A foreign key is used to link two tables together by referencing the primary key of the related table.

# CREATE TABLE – Integrity Constraints

**FORMAT:**

- **primary key** ($A_1$, ..., $A_n$ )
- **foreign key** ($A_m$, ..., $A_n$ ) **references** *r*

*Example:*

```
create table instructor (
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department);
```

**primary key** declaration on an attribute automatically ensures **not null**

# Basic Query Structure – SELECT statement

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

# Basic Query Structure – SELECT statement

- A typical SQL query has the form:

  **select** $A_1, A_2, ..., A_n$
  **from** $r_1, r_2, ..., r_m$
  **where** $P$

  - $A_i$ represents an attribute
  - $R_i$ represents a relation
  - $P$ is a predicate.

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

# Basic Query Structure – SELECT statement

- A typical SQL query has the form:

  **select** $A_1, A_2, ..., A_n$
  **from** $r_1, r_2, ..., r_m$
  **where** $P$

  - $A_i$ represents an attribute
  - $R_i$ represents a relation
  - $P$ is a predicate.

- The result of an SQL query is a relation.

# SELECT clause (in practice)

"selection"

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |

# SELECT clause (in practice)

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

"selection" and "projection"

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

| PName | Price | Manufacturer |
|---|---|---|
| SingleTouch | $149.99 | Canon |
| MultiTouch | $203.99 | Hitachi |

# NOTATION

Input Schema

Product(PName, Price, Category, Manfacturer)

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

Answer(PName, Price, Manfacturer)

Output Schema

# DETAILS

- Case insensitive:
  - Same: SELECT  Select  select
  - Same: Product  product
  - Different: ‘Seattle’  ‘seattle’
- Constants:
  - ‘abc’  - yes
  - “abc” – no
- Each clause starts in a new line
- Long statements: Split up in separate indented lines

# Example:

```
SELECT *
FROM   departments;
```

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|---|---|---|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |
| 60 | IT | 103 | 1400 |
| 80 | Sales | 149 | 2500 |
| 90 | Executive | 100 | 1700 |
| 110 | Accounting | 205 | 1700 |
| 190 | Contracting | | 1700 |

8 rows selected.

Use SELECT to display ALL columns of the relation (tables).
Keyword = *

# Example:

```
SELECT department_id, location_id
FROM   departments;
```

| DEPARTMENT_ID | LOCATION_ID |
|---|---|
| 10 | 1700 |
| 20 | 1800 |
| 50 | 1500 |
| 60 | 1400 |
| 80 | 2500 |
| 90 | 1700 |
| 110 | 1700 |
| 190 | 1700 |

8 rows selected.

Use SELECT to display specific columns of the relation (tables). Specify the column names separated by commas

# Arithmetic Expressions – Add New columns

Create expressions on NUMBER and DATE data with arithmetic operators:

| Operator | Description |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |

Use arithmetic operators in any clause of a SQL statement except the FROM clause.

# Arithmetic Expressions – Add New columns

PROPERTIES:

| Operator | Description |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritised evaluation and to clarify statements.

# Example:

```
SELECT department_id, location_id
FROM   departments;
```

| DEPARTMENT_ID | LOCATION_ID |
|---|---|
| 10 | 1700 |
| 20 | 1800 |
| 50 | 1500 |
| 60 | 1400 |
| 80 | 2500 |
| 90 | 1700 |
| 110 | 1700 |
| 190 | 1700 |

8 rows selected.

Use SELECT to display specific columns of the relation (tables). Specify the column names separated by commas

# Example:

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

| LAST_NAME | SALARY | SALARY+300 |
|---|---|---|
| King | 24000 | 24300 |
| Kochhar | 17000 | 17300 |
| De Haan | 17000 | 17300 |
| Hunold | 9000 | 9300 |
| Ernst | 6000 | 6300 |

...

20 rows selected.

# Example (operator precendence):

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

| LAST_NAME | SALARY | 12*SALARY+100 |
|---|---|---|
| King | 24000 | 288100 |
| Kochhar | 17000 | 204100 |
| De Haan | 17000 | 204100 |

...

20 rows selected.

## Using Parentheses

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2

| LAST_NAME | SALARY | 12*(SALARY+100) |
|---|---|---|
| King | 24000 | 289200 |
| Kochhar | 17000 | 205200 |
| De Haan | 17000 | 205200 |

...

20 rows selected.

# NULL operator in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if can be null (*nullable* attribute) or not

# NULL operator in SQL

- How does SQL cope with tables that have NULLs ?

ANSWER:

If x= NULL then 4*(3-x)/7 is still NULL

If x= NULL then x="Joe"  is UNKNOWN

In SQL there are three boolean values:

| | | |
|---|---|---|
| FALSE | = | 0 |
| UNKNOWN | = | 0.5 |
| TRUE | = | 1 |

More… next lecture

# Example - NULL operator

– A null is a value that is to unavailable, unassigned, unknown, or inapplicable data

– A null is not the same as a zero or a blank space.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

| LAST_NAME | JOB_ID | SALARY | COMMISSION_PCT |
|---|---|---|---|
| King | AD_PRES | 24000 | |
| Kochhar | AD_VP | 17000 | |
| ... | | | |
| Zlotkey | SA_MAN | 10500 | .2 |
| Abel | SA_REP | 11000 | .3 |
| Taylor | SA_REP | 8600 | .2 |
| ... | | | |
| Gietz | AC_ACCOUNT | 8300 | |

20 rows selected.

# Example - NULL operator

– Arithmetic Operations containing NULL result in NULL.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

| LAST_NAME | 12*SALARY*COMMISSION_PCT |
|---|---|
| King | |
| Kochhar | |
| ... | |
| Zlotkey | 25200 |
| Abel | 39600 |
| Taylor | 20640 |
| ... | |
| Gietz | |

20 rows selected.

# Column Alias

A column alias 'renames' a column heading

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

| NAME | COMM |
|---|---|
| King | |
| Kochhar | |
| De Haan | |

...

20 rows selected.

# Column Alias

A column alias 'renames' a column heading

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

| Name | Annual Salary |
|---|---|
| King | 288000 |
| Kochhar | 204000 |
| De Haan | 204000 |

...

If the alias has a special character (e.g., $, blank space), then it has to be enclosed within double quotation marks.

Requires double quotation marks if it contains spaces or special characters or is case sensitive.

# Concatenation Operator

- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, literal character strings, etc.

Creates a new column

# Concatenation Operator

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, literal character strings, etc.

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```

| Employees |
| --- |
| KingAD_PRES |
| KochharAD_VP |
| De HaanAD_VP |

...

20 rows selected.

Creates a new column

# Literal Character Strings (LCS)

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, literal character strings, etc.

An LCS is any character, expression, or number included in the SELECT list added to the column name or a column alias.

In your SLQ code, this string must be enclosed within single quotations marks

# Example:

- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, literal character strings, etc.

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

| Employee Details |
|---|
| King is a AD_PRES |
| Kochhar is a AD_VP |
| De Haan is a AD_VP |
| Hunold is a IT_PROG |
| Ernst is a IT_PROG |
| Lorentz is a IT_PROG |
| Mourgos is a ST_MAN |
| Rajs is a ST_CLERK |

...

# IMPORTANT

## (a) Duplicate rows

The default display of queries includes duplicate rows (all)

```
SELECT department_id
FROM   employees;
```

1

| DEPARTMENT_ID |
| --- |
| 90 |
| 90 |
| 90 |

...

20 rows selected.

# IMPORTANT

## (a) Duplicate rows

The SELECT *DISTINCT* statement is used to return only *distinct* (different) values

```
SELECT DISTINCT department_id
FROM   employees;
```

2

| DEPARTMENT_ID |
|---|
| 10 |
| 20 |
| 50 |

...

# The WHERE clause

Add restrictions to tuple-selection with the WHERE clause

This statement follows the FROM clause and then ';'

```
SELECT        [DISTINCT] {*| column [alias], ...}
FROM          table
[WHERE        condition(s)];
```

# Example:

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|---|---|---|
| 100 | King | AD_PRES | 90 |
| 101 | Kochhar | AD_VP | 90 |
| 102 | De Haan | AD_VP | 90 |

# Example:

```
SELECT last_name, job_id, department_no
FROM   employees
WHERE job_id ='CLERK';
```

Character strings enclosed within single quotation marks

| ENAME | JOB | DEPTNO |
|---|---|---|
| JAMES | CLERK | 30 |
| SMITH | CLERK | 20 |
| ADAMS | CLERK | 20 |
| MILLER | CLERK | 10 |

More… next lecture.

# Display Table Structure

Use DESCRIBE to display the internal structure of a table

```
DESC[RIBE] tablename
```

## Example:

```
DESCRIBE employees
```

| Name | Null? | Type |
|---|---|---|
| EMPLOYEE_ID | NOT NULL | NUMBER(6) |
| FIRST_NAME | | VARCHAR2(20) |
| LAST_NAME | NOT NULL | VARCHAR2(25) |
| EMAIL | NOT NULL | VARCHAR2(25) |
| PHONE_NUMBER | | VARCHAR2(20) |
| HIRE_DATE | NOT NULL | DATE |
| JOB_ID | NOT NULL | VARCHAR2(10) |
| SALARY | | NUMBER(8,2) |
| COMMISSION_PCT | | NUMBER(2,2) |
| MANAGER_ID | | NUMBER(6) |
| DEPARTMENT_ID | | NUMBER(4) |

Any questions?

Readings: See CANVAS

# References:

(a) A Conceptual Poverty Mapping Data Model
Link: https://www.researchgate.net/figure/Key-thematic-layers-for-poverty-spatial-data-modeling_fig2_229724703

(b) Relational Database relationships
https://www.youtube.com/watch?v=C3icLzBtg8I

(c) https://courses.ischool.berkeley.edu/i202/f97/Lecture13/DatabaseDesign/sld002.htm

(d) https://nexwebsites.com/database/database-management-systems/

(e) Acknowledgement – Thanks to http://courses.cs.washington.edu/courses/cse544/ for providing part of this presentation.

(f) Acknowledgement – Thanks to © Silberchatz, Korth and Surdashan for providing part of this presentation.

(e) Malinowski, Elzbieta, Zimányi, Esteban (2008) *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications* . Springer Berlin Heidelberg