ON THE SYLLABUS

Evaluation and Grading

Must be in TEAMS to take any graded assignment. Any attempt to take a graded assignment not being in TEAMS will result in a grade of ZERO for that graded assignment.

Webcams are required to take EXAMs.

30% zyBook
20% Class Participation
50% EXAMs (4)

Grading Scale:

93 or higher A
90 – below 93 A-
87– below 90 B+
83 – below 87 B
80 – below 83 B-
77 – below 80 C+
73 – below 77 C
70 – below 73 C-
67 – below 70 D+
63 – below 67 D
60 – below 63 D-
0 – below 60 F

Search content Search View content explorer Configure zyBook

Showing activity for entire class zyLabs Challenge Participation

	zyLabs	Challenge	Participation
1. SET 1			Empty
2. SET 1 - 1:INTRODUCTION			100%
3. SET 1 - 2:DATA MODELING - WHAT - ERD MODEL		98%	99%
4. SET 1 - 3:DATA MODELING HOW - RELATIONAL MO...		97%	98%
5. SET 1 - 4:ERD to RELATIONAL		94%	99%
6. SET 1 - 5:NORMALIZATION	2%	95%	98%
7. SET 2			Empty
8. SET 2 - 1:SQL I	0%	6%	8%

COSC 3380: Database Systems

Spring 2024

View activity and create a report

1. Select chapters and sections in the table of contents.
2. Then select class and time options below.

Entire class

From: Select date Select time CST

Until: Feb 14th, 2024 11:59 pm CST

Stop viewing entire class

All activity up until Feb 14th, 2024 at 11:59 pm CST will be downloaded.

Include data on time spent in chapters, sections, and on activities

Download report

You must select at least one section from the table of contents to

UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

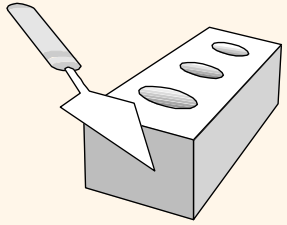
Database Systems

M & W 4:00 to 5:30 PM

Prof. Victoria Hilford

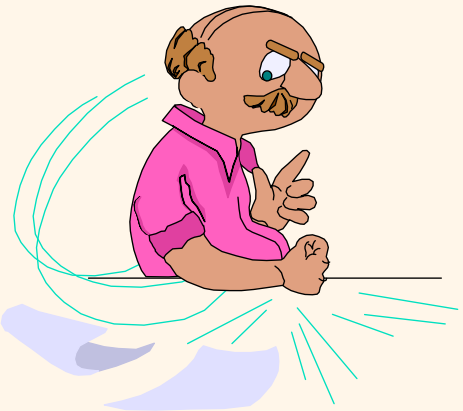
PLEASE TURN your webcam ON (must have)

NO CHATTING during LECTURE



COSC 3380

4 to 5:30

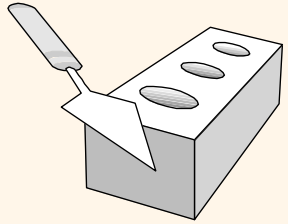


PLEASE
LOG IN
CANVAS

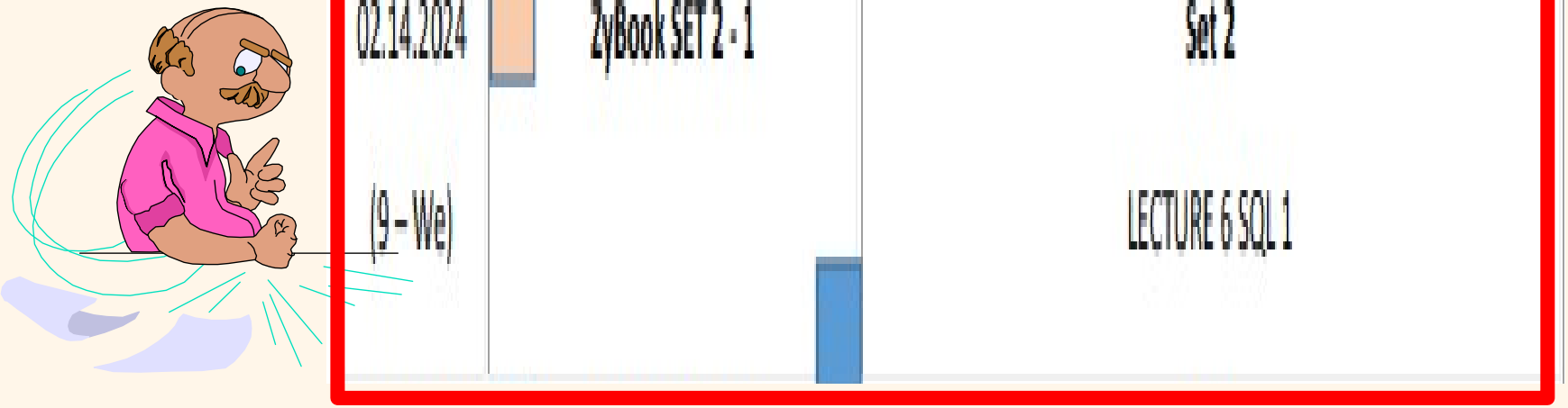
Please close all other windows.

02.14.2024 (9 – We)	**ZyBook SET 2 - 1**	**Set 2** LECTURE 6 SQL 1
02.19.2024 (10 - Mo)	**ZyBook SET 2 - 2**	**Set 2** LECTURE 7 SQL II
02.21.2024 (11 - We)	**ZyBook SET 2- 3**	**Set 2** LECTURE 10 APPLICATIONS
02.26.2024 (12 - Mo)	**ZyBook SET 2 - 4**	**Set 2** LECTURE 11 WEB APPLICATIONS
02.28.2024 (13 - We)		**EXAM 2 Practice** **(PART of 20 points)**
03.04.2024 (14 - Mo)	**TA Download** **ZyBook SET 2 Sections** **(4 PM)** **(PART of 30 points)**	**EXAM 2 Review** **(PART of 20 points)**
03.06.2024 (15 - We)		**EXAM 2** **(PART of 50 points)**

COSC 3380



Class 9



From 4:00 to 4:07 PM – 5 minutes.

02.14.2024 (9 – We)	ZyBook SET 2 - 1	Set 2 LECTURE 6 SQL 1

CLASS PARTICIPATION 20 points 20% of Total

HAPPY VALENTINE

Class 9 BEGIN PARTICIPATION
Not available until Feb 14 at 4:00pm | Due Feb 14 at 4:07pm | 100 pts

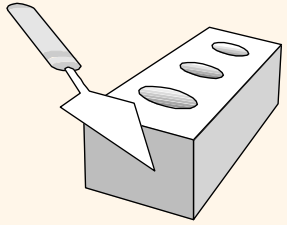
VH, publish

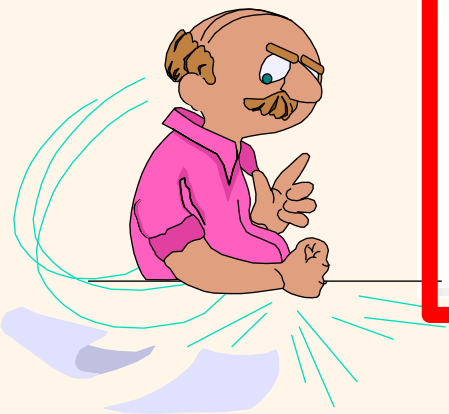
This is a synchronous online class.

Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

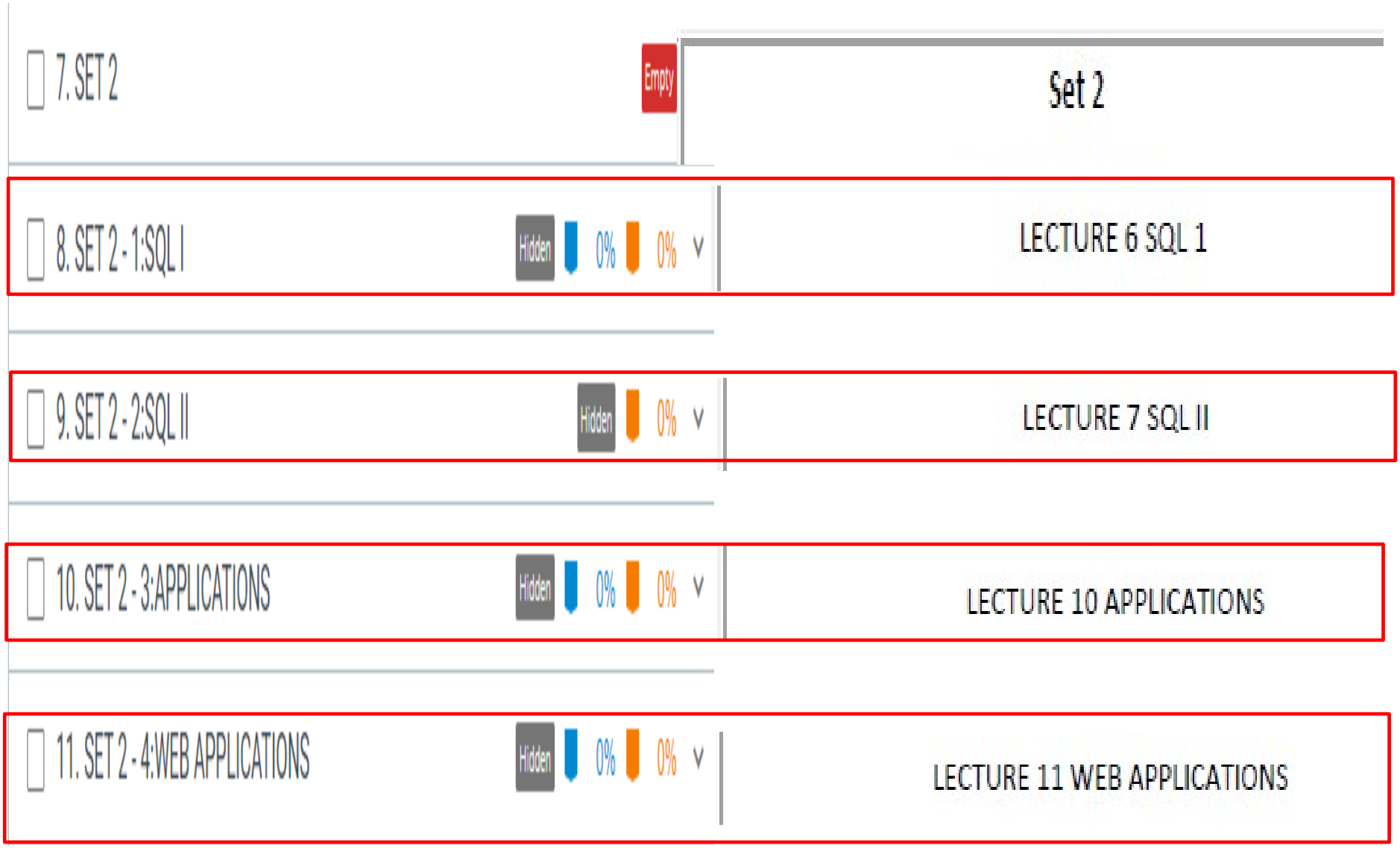




02.14.2024	ZyBook SET 2 - 1	Set 2
(9 - We)		LECTURE 6 SQL 1

ata

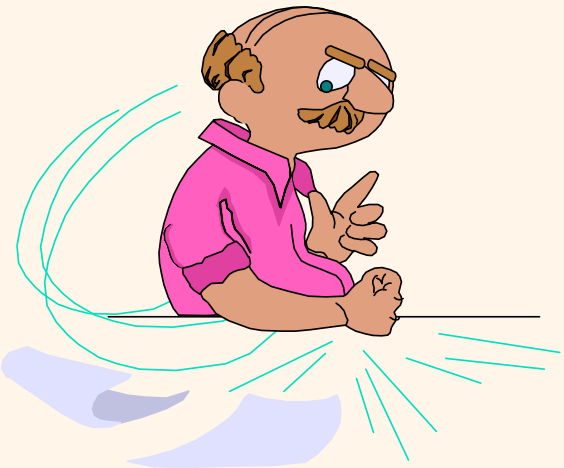
Mapping ZyBook SET 2 to the Lectures



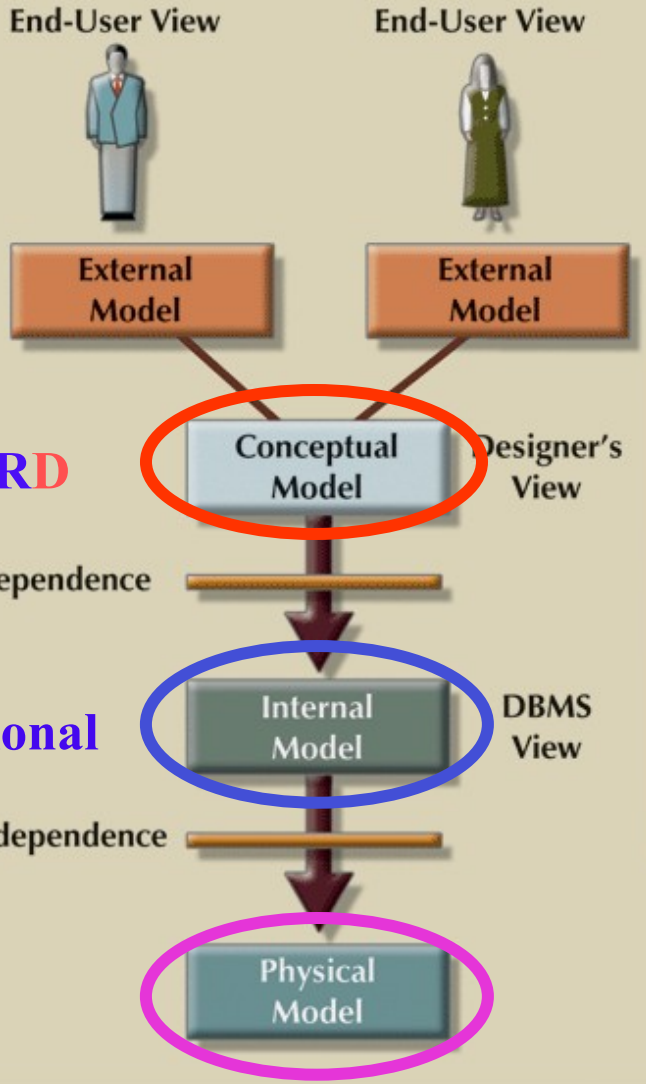
From 4:07 to 5:00 PM – 53 minutes.

Lecture 6

SQL: Tables and Queries



Data abstraction levels



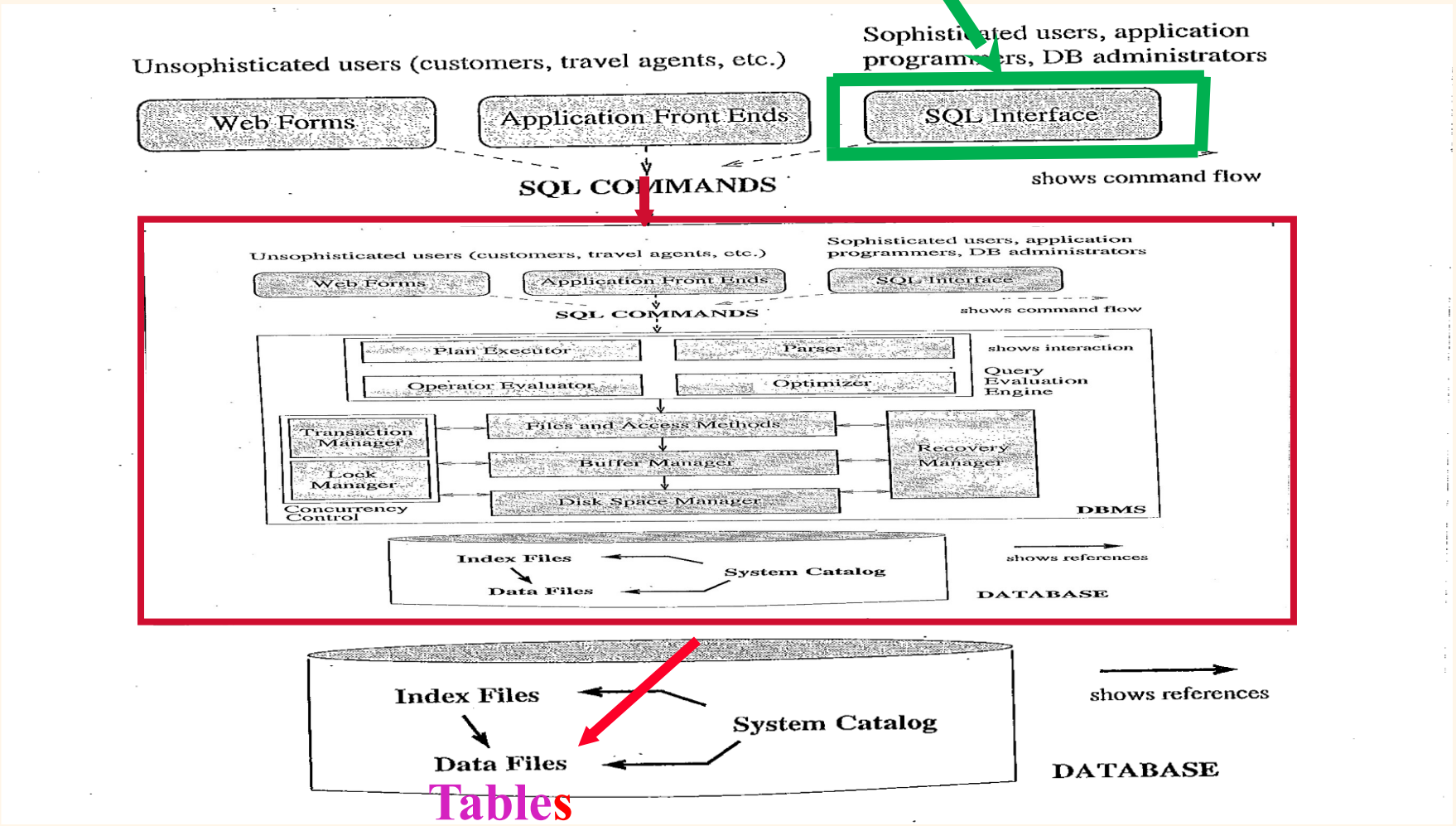
What - ERD

How - Relational

SQL

Degree of Abstraction		Characteristics
High	ER	Hardware-independent Software-independent
	Object-Oriented	
Medium	Relational	Hardware-independent Software-dependent
Low	Network Hierarchical	Hardware-dependent Software-dependent

SQL – Structured Query Language



21.1 MySQL

MySQL

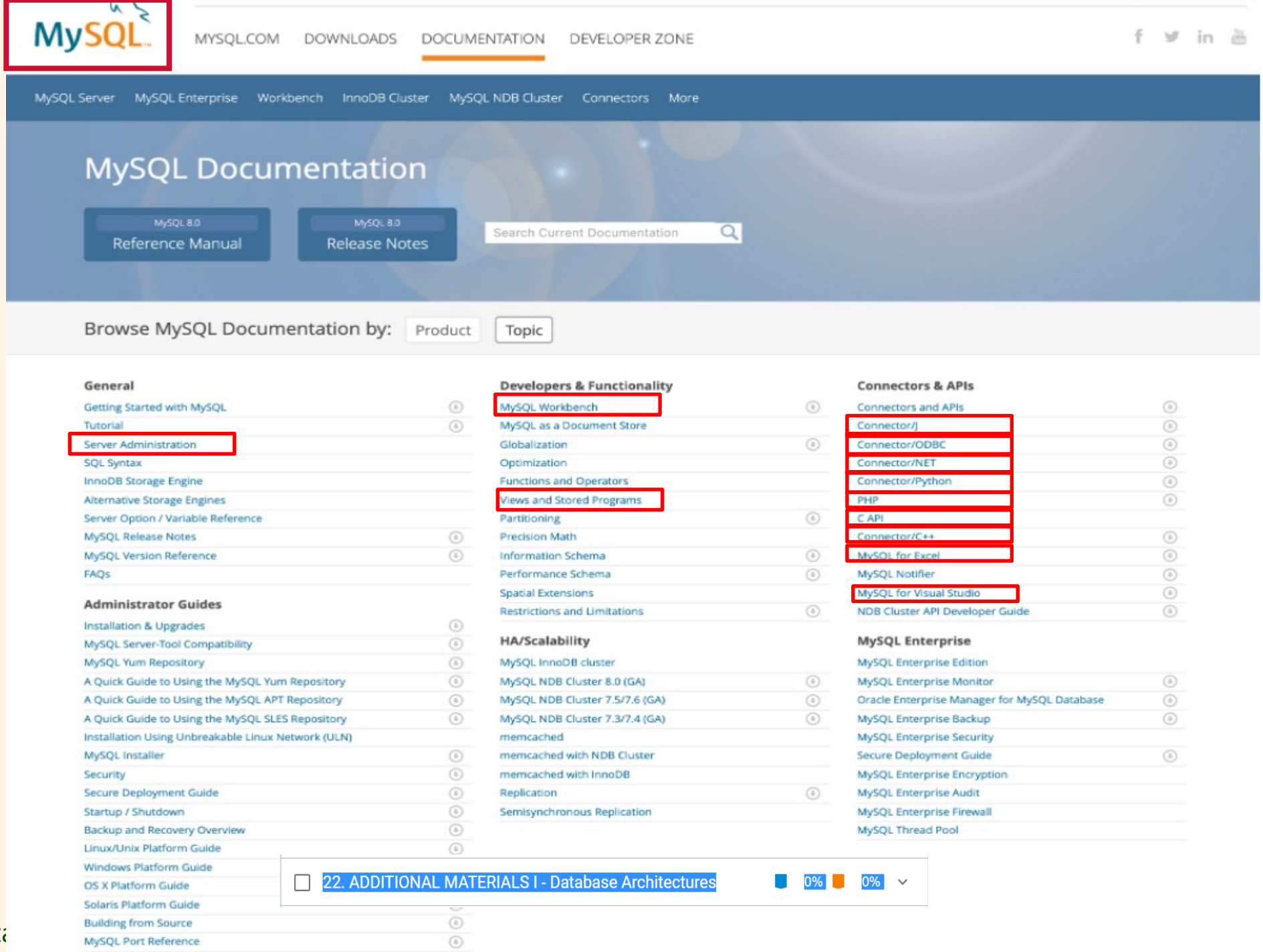
This material uses MySQL as a reference relational database system. Although the material is relevant to all relational databases, SQL syntax and many activities are based on MySQL Server release 8.0.

MySQL is a leading relational database system sponsored by Oracle. MySQL is relatively easy to install and use, yet has many advanced capabilities. MySQL runs on all major operating systems, including Linux, Unix, Mac OS, and Windows. For these reasons, MySQL is one of the most popular database systems.

The latest MySQL release is 8.0, available in two editions:

- ***MySQL Community***, commonly called ***MySQL Server***, is a free edition. MySQL Server includes a complete set of database services and tools, and is suitable for non-commercial applications such as education.
- ***MySQL Enterprise*** is a paid edition for managing commercial databases. MySQL Enterprise includes MySQL Server and additional administrative applications.

Complete documentation for MySQL Server 8.0 is available online.


MySQL
MYSQL.COM
DOWNLOADS
DOCUMENTATION
DEVELOPER ZONE
MySQL Server
MySQL Enterprise
Workbench
InnoDB Cluster
MySQL NDB Cluster
Connectors
More
MySQL Documentation
MySQL 8.0
Reference Manual
MySQL 8.0
Release Notes
Search Current Documentation
Browse MySQL Documentation by:
Product
Topic
General
Getting Started with MySQL
Tutorial
Server Administration
SQL Syntax
InnoDB Storage Engine
Alternative Storage Engines
Server Option / Variable Reference
MySQL Release Notes
MySQL Version Reference
FAQs
Administrator Guides
Installation & Upgrades
MySQL Server-Tool Compatibility
MySQL Yum Repository
A Quick Guide to Using the MySQL Yum Repository
A Quick Guide to Using the MySQL APT Repository
A Quick Guide to Using the MySQL SLES Repository
Installation Using Unbreakable Linux Network (ULN)
MySQL Installer
Security
Secure Deployment Guide
Startup / Shutdown
Backup and Recovery Overview
Linux/Unix Platform Guide
Windows Platform Guide
OS X Platform Guide
Solaris Platform Guide
Building from Source
MySQL Port Reference
Developers & Functionality
MySQL Workbench
MySQL as a Document Store
Globalization
Optimization
Functions and Operators
Views and Stored Programs
Partitioning
Precision Math
Information Schema
Performance Schema
Spatial Extensions
Restrictions and Limitations
HA/Scalability
MySQL InnoDB cluster
MySQL NDB Cluster 8.0 (GA)
MySQL NDB Cluster 7.5/7.6 (GA)
MySQL NDB Cluster 7.3/7.4 (GA)
memcached
memcached with NDB Cluster
memcached with InnoDB
Replication
Semisynchronous Replication
Connectors & APIs
Connectors and APIs
Connector/J
Connector/ODBC
Connector/NET
Connector/Python
PHP
C API
Connector/C++
MySQL for Excel
MySQL Notifier
MySQL for Visual Studio
NDB Cluster API Developer Guide
MySQL Enterprise
MySQL Enterprise Edition
MySQL Enterprise Monitor
Oracle Enterprise Manager for MySQL Database
MySQL Enterprise Backup
MySQL Enterprise Security
Secure Deployment Guide
MySQL Enterprise Encryption
MySQL Enterprise Audit
MySQL Enterprise Firewall
MySQL Thread Pool
22. ADDITIONAL MATERIALS I - Database Architectures
0%
0%

MySQL Command-Line Client

The ***MySQL Command-Line Client*** is a text interface included in the MySQL Server download. The Command-Line Client allows developers to connect to the database server, perform administrative functions, and execute SQL statements.

To run the Command-Line Client, a user must first open a Command Prompt on Windows or a Terminal on a Mac:

- Windows: Click the Start button in the Taskbar, type "cmd", then click Command Prompt.
- Mac: Click on the Terminal application, usually found in the Applications > Utilities folder.

When MySQL Command-Line Client is started with the root account, the user is prompted to enter the root account password. Then Command-Line Client attempts to connect to the database server running on the local machine.

PARTICIPATION ACTIVITY 21.1.2: Using the MySQL Command-Line Client.

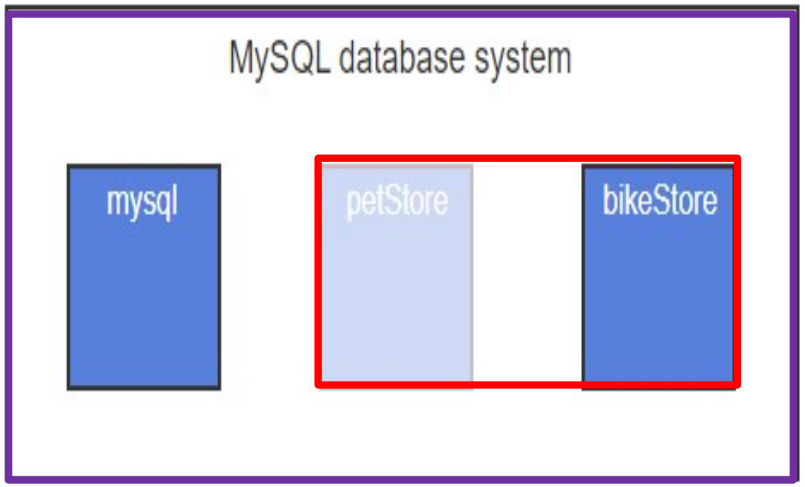
```
$ mysql -u root -p
Enter password: ********
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
...
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE world;
Database changed.
mysql> SELECT * FROM city LIMIT 10;
+----+----------------+-------------+---------------+------------+
| ID | Name           | CountryCode | District      | Population |
+----+----------------+-------------+---------------+------------+
|  1 | Kabul          | AFG         | Kabol         |    1780000 |
|  2 | Qandahar       | AFG         | Qandahar      |     237500 |
|  3 | Herat          | AFG         | Herat         |     186800 |
|  4 | Mazar-e-Sharif | AFG         | Balkh         |     127800 |
|  5 | Amsterdam      | NLD         | Noord-Holland |     731200 |
|  6 | Rotterdam      | NLD         | Zuid-Holland  |     593321 |
|  7 | Haag           | NLD         | Zuid-Holland  |     440900 |
|  8 | Utrecht        | NLD         | Utrecht       |     234323 |
|  9 | Eindhoven      | NLD         | Noord-Brabant |     201843 |
| 10 | Tilburg        | NLD         | Noord-Brabant |     193238 |
+----+----------------+-------------+---------------+------------+
10 rows in set (0.00 sec)

mysql>
```

- [ ] 8. SET 2 - 1:SQL I — 0% 0%
  - [ ] 8.1 Inner and outer joins — 0% 0%
  - [ ] 8.2 View tables — 0% 0%
  - [ ] 8.3 Relational algebra — 0% 0%
  - [ ] 8.4 Query languages — 0% 0%
  - [ ] 8.5 Introduction to SQL — 0%
  - [ ] 8.6 Creating and dropping databases — 0%
  - [ ] 8.7 Creating and dropping tables — 0% 0%
  - [ ] 8.8 Primary and foreign key constraints — 0% 0%
  - [ ] 8.9 Column constraints — 0% 0%
  - [ ] 8.10 Inserting, updating, and deleting rows — 0% 0%
  - [ ] 8.11 Selecting rows — 0% 0%
  - [ ] 8.12 Selecting rows: Additional capabilities — 0% 0%

CREATE DATABASE and DROP DATABASE statements

The ***CREATE DATABASE*** statement creates a new database. Once a database is created, tables can be added to the database. The ***DROP DATABASE*** statement deletes the database, including all tables in the database.

PARTICIPATION ACTIVITY | 8.6.1: CREATE DATABASE and DROP DATABASE.




```
CREATE DATABASE petStore;
CREATE DATABASE bikeStore;
DROP DATABASE petStore;
```

Creating Tables using SQL

Lecture 6 SQL.sql

Students(sid:string,name:string,login:string,age:integer,gpa:real)

```
DROP TABLE Students CASCADE CONSTRAINTS;
```

```
CREATE TABLE Students
        (sid    CHAR(20) PRIMARY KEY,
         name   CHAR(20),
         login  CHAR(10),
         age    INTEGER,
         gpa    REAL);
```

```
/* Slide 8 */

DROP TABLE Students CASCADE CONSTRAINTS;

CREATE TABLE Students(sid  CHAR(20) PRIMARY KEY, name CHAR(20), login CHAR(10),
age INTEGER, gpa REAL);

DESCRIBE Students;
```

sid	name	login	age	gpa

ANSI Standard	Oracle Datatype
CHARACTER and CHAR	CHAR
CHARACTER VARYING and CHAR VARYING	VARCHAR2
NUMERIC, DECIMAL, DEC, INTEGER, INT and SMALLINT	INTEGER
FLOAT, REAL, DOUBLE PRECISION	REAL

```
DESCRIBE Students;
```

Table 8.7.1: Common MySQL data types.

Category	Example	Data type	Storage	Notes
Integer	34 and -739448	TINYINT	1 byte	Signed range: -128 to 128 Unsigned range: 0 to 255
		SMALLINT	2 bytes	Signed range: -32,768 to 32,767 Unsigned range: 0 to 65,535
		MEDIUMINT	3 bytes	Signed range: -8,388,608 to 8,388,607 Unsigned range: 0 to 16,777,215
		INTEGER or INT	4 bytes	Signed range: -2,147,483,648 to 2,147,483,647 Unsigned range: 0 to 4,294,967,295
		BIGINT	8 bytes	Signed range: -2^{63} to $2^{63}-1$ Unsigned range: 0 to $2^{64}-1$
Decimal	123.4 and -54.29685	DECIMAL(M,D)	Varies depending on *M* and *D*	Exact decimal number where *M* = number of significant digits, *D* = number of digits after decimal point
		FLOAT	4 bytes	Approximate decimal numbers with range: -3.4E+38 to 3.4E+38
		DOUBLE	8 bytes	Approximate decimal numbers with range: -1.8E+308 to 1.8E+308
Date and time	'1776-07-04 13:45:22'	DATE	3 bytes	Format: YYYY-MM-DD. Range: '1000-01-01' to '9999-12-31'
		TIME	3 bytes	Format: hh:mm:ss
		DATETIME	5 bytes	Format: YYYY-MM-DD hh:mm:ss. Range: '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
Character	'string'	CHAR(N)	*N* bytes	Fixed-length string of length *N*; $0 \le N \le 255$
		VARCHAR(N)	Length of characters + 1 bytes	Variable-length string with maximum *N* characters; $0 \le N \le 65,535$

Creating tables with SQL

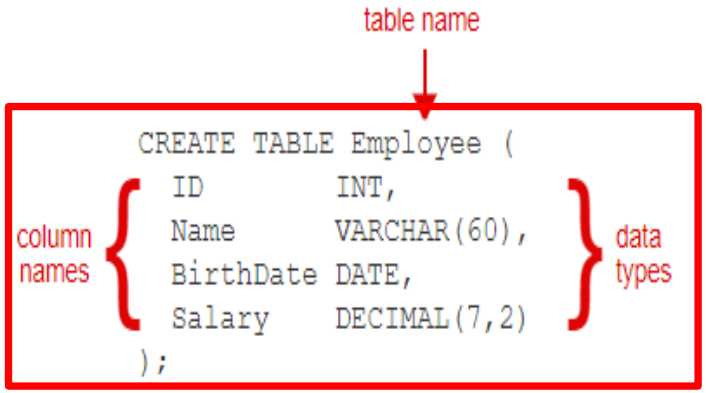
The SQL ***CREATE TABLE*** statement creates a new table by specifying the table and column names. Each column is assigned a ***data type*** that indicates the format of column values. Data types can be numeric, textual, or complex. Ex:

- INT stores integer values.
- DECIMAL stores fractional numeric values.
- VARCHAR stores textual values.
- DATE stores year, month, and day.

Some data types are followed by one or two numbers in parentheses, indicating the size of the data type. Ex: VARCHAR(10) indicates ten characters. DECIMAL(10, 3) indicates ten significant digits, including three after the decimal point.

PARTICIPATION ACTIVITY 8.4.5: Creating an Employee table.

PRIMARY KEY please



Employee

ID	Name	BirthDate	Salary

Modifying *Tables* using SQ

sid	name	login	age	gpa

```
INSERT
INTO Students
VALUES ('53666', 'Jones', 'jones@cs',18,
3.4);
```

```
INSERT
INTO Students
VALUES ('53688', 'Smith', 'smith@eecs',18,
3.2) ;
```

```
INSERT
INTO Students
VALUES ('53650', 'Smith',
'smith@math',19, 3.8) ;
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Students

```
/* Slide 9 */

INSERT INTO Students VALUES ( '53666', 'Jones', 'jones@cs',  18, 3.4);
INSERT INTO Students VALUES ( '53688', 'Smith', 'smith@eecs', 18, 3.2);
INSERT INTO Students VALUES ( '53650', 'Smith', 'smith@math', 19, 3.8);

SELECT * FROM Students S;
```

Modifying *Tables* using SQL

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

```
DELETE
FROM Students S
WHERE S.name = 'Smith' ;
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4

```
SQL> /* Slide 10 */
SQL>
SQL> DELETE FROM Students S WHERE (S.name = 'Smith');

2 rows deleted.

SQL>
SQL> SELECT * FROM Students;

SID                  NAME                 LOGIN          AGE          GPA
-------------------- -------------------- ---------- ---------- ----------
53666                Jones                jones@cs           18          3.4
```

Modifying *Tables* using SQL

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4

```
UPDATE  Students S
SET S.gpa = S.gpa – 0.1
WHERE S.gpa >=3.3;
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.3

```
SQL> /* Slide 11 */
SQL>
SQL> UPDATE Students S SET s.gpa = S.gpa -0.1 WHERE S.gpa >= 3.3;

1 row updated.

SQL>
SQL> SELECT * FROM Students S;

SID                  NAME                 LOGIN                AGE        GPA
-------------------- -------------------- ---------- ---------- ----------
53666                Jones                jones@cs             18        3.3
```

Writing queries with SQL

Structured Query Language, or ***SQL***, is the standard query language of relational database systems. The SQL standard is sponsored by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). SQL is pronounced either 'S-Q-L' or 'seekwəl', but the preferred pronunciation is 'S-Q-L'.

SQL was first developed at IBM in the 1970s as an experimental query language for a prototype relational database. At the time, IBM was the dominant computer company, so SQL became the dominant relational query language. Today, all relational database systems support SQL.

Terminology

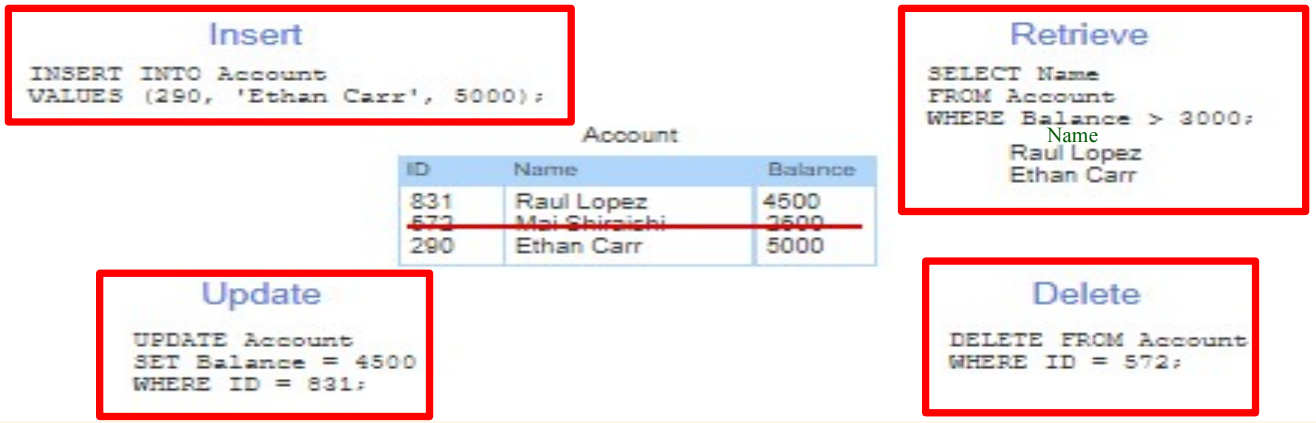
*The term **NoSQL** refers to a new generation of non-relational databases. NoSQL originally meant 'does not support SQL'. However, many NoSQL databases have added support for SQL, and 'NoSQL' has come to mean 'not only SQL'.*

An SQL ***statement*** is a database command, such as a query that inserts, retrieves, updates, or deletes data:

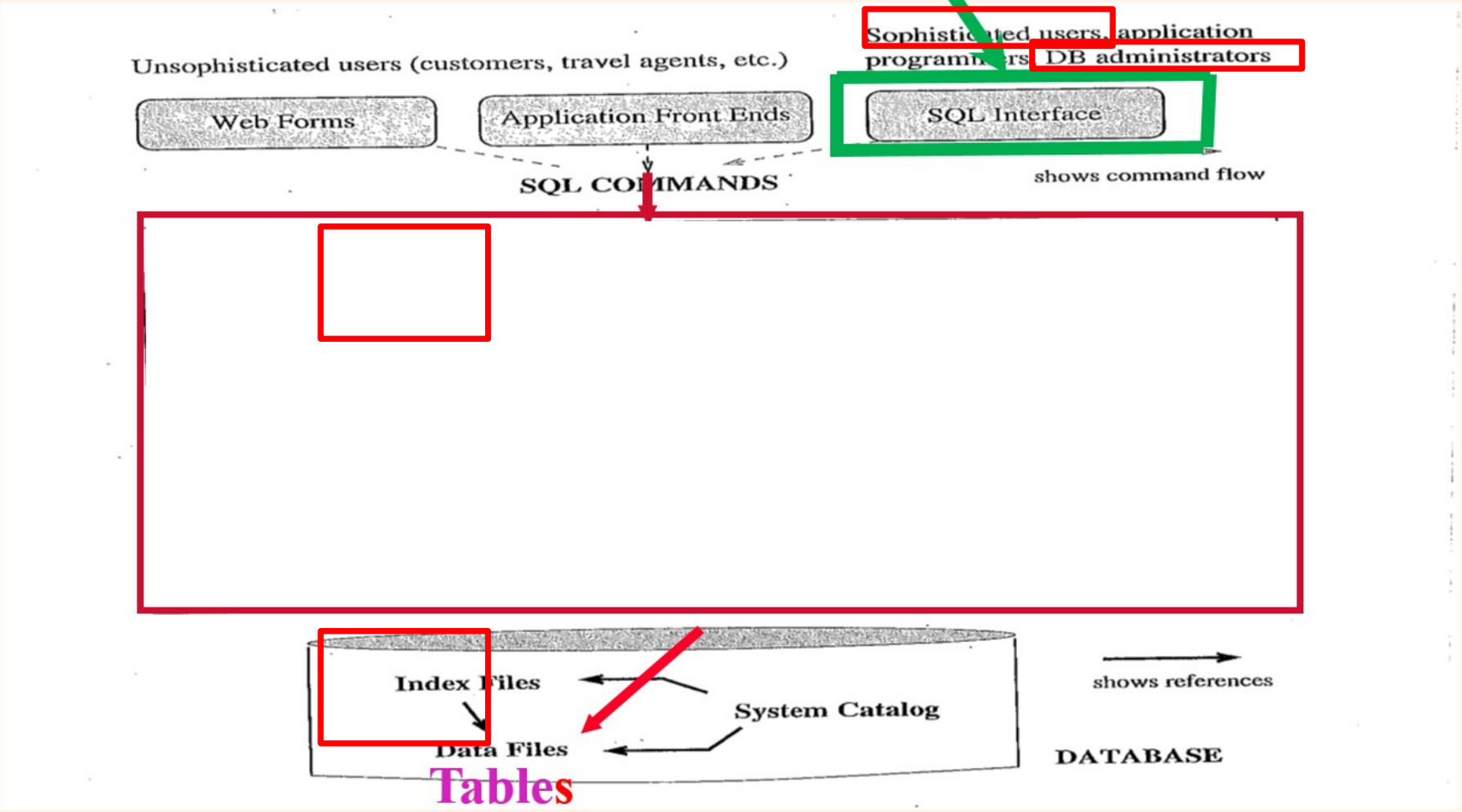
- ***INSERT*** inserts rows into a table.
- ***SELECT*** retrieves data from a table.
- ***UPDATE*** modifies data in a table.
- ***DELETE*** deletes rows from a table.

The SQL language contains many other statements for creating and deleting databases, creating and deleting tables, assigning user permissions, and so on.

PARTICIPATION ACTIVITY 8.4.3: SQL statements: INSERT, SELECT, UPDATE, and DELETE.



SQL – Structured Query Language



Querying *Tables* using SQL

To **find all 18 year old students**, we can write:

?????

```
SELECT *
FROM Students S
WHERE S.age=18;
```

Write the SQL!

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

S

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

To find just names and logins, replace the first

Write the output Relation!

```
SQL>
SQL> SELECT * FROM Students S WHERE s.age =18;

SID                  NAME                 LOGIN                AGE        GPA
-------------------- -------------------- ---------- ---------- ----------
53666                Jones                jones@cs             18        3.4
53688                Smith                smith@eecs           18        3.2

SQL>
SQL> SELECT S.name, S.login FROM Students S WHERE s.age =18;

NAME                 LOGIN
-------------------- ----------
Jones                jones@cs
Smith                smith@eecs
```

Querying Multiple Tables

- What does the following query compute?

```
SELECT  S.name, E.cid
FROM  Students S, Enrolled E
WHERE  S.sid=E.sid AND E.grade='A';
```

Given the following instances S of **Students** & E of **Enrolled**

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

S

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

E

we get:

name	cid
Smith	Topology112

```
SQL>
SQL> /*  Slide 14 */
SQL>
SQL> SELECT S.name, E.cid FROM Students S, Enrolled E WHERE S.sid = E.sid AND E.grade = 'A';

NAME                 CID
-------------------- --------------------
Smith                Topology112
```

Destroying and Altering Tables in SQL

```
DROP TABLE Students;
```

- Destroys the **Relation**/**Table** tuples are deleted.

```
ALTER TABLE Students
        ADD firstYear integer;
```

- The schema of **Students** is altered by adding a new field; every tuple in the current instance is extended with a ***null*** value in the new field.

```
/*  Slide 15 */
ALTER TABLE Students ADD firstYear INTEGER;
SELECT * FROM Students S;
```

PRIMARY KEY constraint

A ***constraint*** is a rule that applies to table data. Constraints are specified in a CREATE TABLE statement or may be added to a preexisting table with an ALTER TABLE statement.

The ***PRIMARY KEY*** constraint in a CREATE TABLE statement names the table's primary key, the column(s) that uniquely identify each row.

PARTICIPATION ACTIVITY 8.8.1: Adding primary key constraints to tables.

```
CREATE TABLE Employee (
   ID         SMALLINT UNSIGNED,
   Name       VARCHAR(60),
   Salary     DECIMAL(7,2),
   PRIMARY KEY (ID)
);
```

Employee

ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30400
6381	Maria Rodriguez	92300

```
CREATE TABLE Family (
   ID           SMALLINT UNSIGNED,
   Number       SMALLINT UNSIGNED,
   Relationship VARCHAR(20),
   Name         VARCHAR(60),
   PRIMARY KEY(ID, Number)
);
```

Family

(ID	Number)	Relationship	Name
2538	1	Spouse	Henry Ellison
2538	2	Son	Edward Ellison
6381	1	Spouse	Jose Rodriguez
6381	2	Daughter	Gina Rodriguez
6381	3	Daughter	Clara Rodriguez

Referential Integrity in SQL

SQL/92 and **SQL**/99 support all 4 options on **deletes** and **updates**.

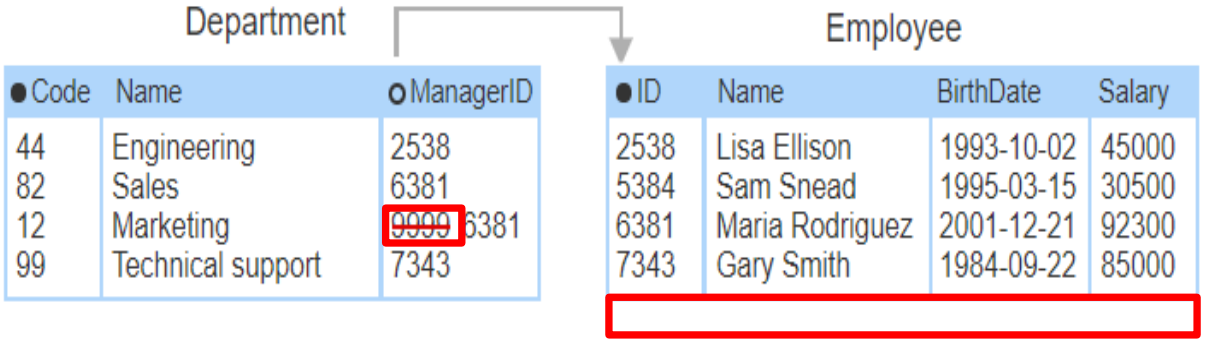
- Default is NO ACTION (*delete/update is rejected*)
- CASCADE (also **delete** all tuples that refer to **deleted** tuple)
- SET NULL / SET DEFAULT (**sets *Foreign key* value of referencing tuple**)

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid) REFERENCES Students(sid)
          ON DELETE CASCADE
          ON UPDATE SET DEFAULT )
```

FOREIGN KEY constraint

A foreign key constraint is added to a CREATE TABLE statement with the ***FOREIGN KEY*** and ***REFERENCES*** keywords. The CREATE statement in the animation below indicates the foreign key ManagerID column refers to the primary key ID in the Employee table. When a foreign key constraint is specified, the database will not allow an insert or update that violates referential integrity. Ex: Inserting a row in Department with ManagerID 9999 is rejected if Employee ID 9999 does not exist.

PARTICIPATION ACTIVITY 8.8.4: Foreign key constraint on the Department table.




```
CREATE TABLE Department (
   Code      TINYINT UNSIGNED,
   Name      VARCHAR(20),
   ManagerID SMALLINT UNSIGNED,
   PRIMARY KEY (Code),
   FOREIGN KEY (ManagerID) REFERENCES Employee(ID)
);
```

ON DELETE and ON UPDATE actions

Actions can be specified on the FOREIGN KEY constraint with ON DELETE and ON UPDATE keywords:

- ***ON DELETE*** responds to an invalid primary key deletion. Ex: Deleting a primary key 1234 that is used in a foreign key.
- ***ON UPDATE*** responds to an invalid primary key update. Ex: Updating a primary key 1234 to 5555 when 1234 is used in a foreign key.

ON DELETE and ON UPDATE must be followed by a response:

- ***RESTRICT*** rejects an insert, update, or delete that violates referential integrity. RESTRICT is applied by default when no action is specified.
- ***SET NULL*** sets an invalid foreign key value to NULL.
- ***SET DEFAULT*** sets invalid foreign keys to a default primary key value.
- ***CASCADE*** propagates primary key changes to foreign keys. If a primary key is deleted, rows containing matching foreign keys are deleted. If a primary key is updated, matching foreign keys are updated to the same value.

PARTICIPATION ACTIVITY 8.8.7: Foreign key constraints with ON DELETE and ON UPDATE actions.

Department

Code	Name	○ManagerID
44	Engineering	NULL
82	Sales	6381
12	Marketing	6381
~~99~~	~~Technical support~~	~~7343~~

Employee

●ID	Name	BirthDate	Salary
8754	Lisa Ellison	1993-10-02	45000
5384	Sam Snead	1995-03-15	30500
6381	Maria Rodriguez	2001-12-21	92300
~~7343~~	~~Gary Smith~~	~~1984-09-22~~	~~85000~~

```
CREATE TABLE Department (
   Code       TINYINT UNSIGNED,
   Name       VARCHAR(20),
   ManagerID SMALLINT UNSIGNED,
   PRIMARY KEY (Code),
   FOREIGN KEY (ManagerID) REFERENCES Employee(ID)
      ON DELETE CASCADE
      ON UPDATE SET NULL
);
```

Views in SQL

Students

SID	NAME	LOGIN	AGE	GPA
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

A ***view*** is just a **Relation**, but we store a *definition*, rather than a set of tuples

Enrolled

SID	CID	GR
53831	Carmatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

```
CREATE VIEW YoungActiveStudents (NAME, GR)
        AS SELECT S.NAME, E.GR
        FROM Students S, Enrolled E
        WHERE S.SID = E.SID and S.AGE < 21;
```

Views can be dropped using the DROP **VIEW** command

- How to handle DROP **TABLE** if there's a **View** on the **Table**
  - DROP **TABLE** command has options to let the user

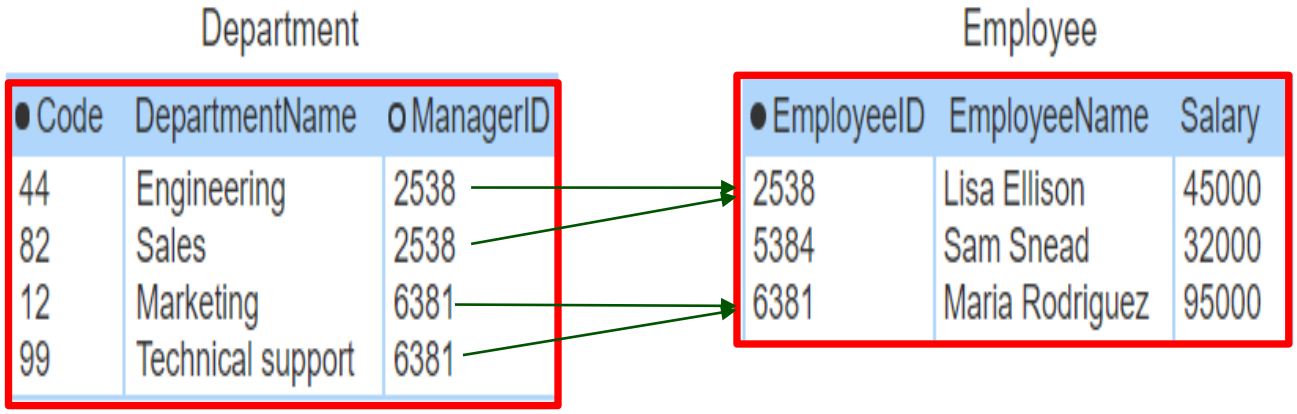
YoungActiveStudents is.

NAME	GR
Smith	A
Jones	B

?????

Welcome to 4GL

PARTICIPATION ACTIVITY 8.2.1: Creating a view table.




```
CREATE VIEW ManagerView
AS SELECT DepartmentName, EmployeeName AS ManagerName
   FROM Department, Employee
   WHERE ManagerID = EmployeeID;
```

view query

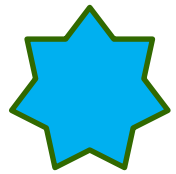
TA time (Fernando) – 3 minutes
(CA 8.6.1 Step 1 –View Tables)

CHALLENGE ACTIVITY | 8.6.1: View tables.

Base table:

Country

Name	Density	ISOCode3	IndependenceYear
Honduras	221	HND	1821
El Salvador	802	SLV	1821
Belgium	977	BEL	1831
Bahamas	99	BHS	1973



View table:

CountryView

Name	Code	IndependenceYear
Honduras	HND	1821
El Salvador	SLV	1821
Belgium	BEL	1831
Bahamas	BHS	1973

Complete the following statement to generate the view table:

```
CREATE VIEW CountryView
AS SELECT Name, ISOCode3 AS Code, IndependenceYear
   FROM Country;
```

(A) CountryView

(B) Name

1

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

42:55

Take control | Pop out | Chat | People 112 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

Arc File Edit View Spaces Tabs Archive Extensions Window Help — Wed Feb 14 4:33:26 PM

learn.zybooks.com

School
- COSC 2425
- COSC 1336 (TA)
- COSC 3320
- COSC 3360
- COSC 3380
- COSC 4393
- COSC 3380 (TA)
  - COSC 3380: D...
  - Section 8.6 -...
  - Section 8.8 -...

New Tab
Class 9 COSC 3380 D...
Calendar

Arc is ready to update!
Click to restart

Ramirez, Fernando

zyBooks My library > COSC 3380: Database Systems home > 8.6: View tables — zyBooks catalog | Help/FAQ | Fernando Ramirez

CHALLENGE ACTIVITY | 8.6.1: View tables.

Jump to level 1

Base table:

Country

Name	Density	Capital	Area
Cuba	282	Havana	40162
Lesotho	179	Maseru	11722
Cambodia	238	Phnom Penh	68154
Netherlands	1324	Amsterdam	13007

View table:

CountryKM2

CountryName	KM2
Cuba	40162
Lesotho	11722
Cambodia	68154
Netherlands	13007

Complete the following statement to generate the view table:

```
CREATE VIEW __(A)__
AS SELECT __(B)__, Area AS KM2
   FROM Country;
```

(A) CountryKM2

(B) Name AS CountryName

1 2 3

Check Next

✔ Expected:

```
CREATE VIEW CountryKM2
AS SELECT Name AS CountryName, Area AS KM2
   FROM Country;
```

`CREATE VIEW CountryKM2` creates a view table named CountryKM2.
`AS SELECT Name AS CountryName` specifies that Name in the base table is renamed CountryName in the view table.

View solution (Instructors only)

Participants

Invite someone or dial a number

Share invite

In this meeting (112) — Mute all

- Hilford, Victoria — Organizer
- Adhikari, Rohit
- Ahmed, Mohamed A
- Akram, Ali
- Altaf, Sameer
- Alvarez, Stephanie
- Anayor-Achu, Ogochukwu E
- Avci, Hatice Kubra
- Aysola, Riya
- Bahl, Anish
- Banza, Sean Paolo B
- Bui, Hieu
- Burger, Jake
- Carrillo-Zepeda, Victor E
- Carter, Deric
- Collier, Rachel
- Dao, Cong Duy Vuong
- Davis, Jacob
- Duong, Vuong
- Elkins, Victor R

See more

Advantages of views

View tables have several advantages:

- **Protect sensitive data**. A table may contain sensitive data. Ex: The Employee table contains compensation columns such as Salary and Bonus. A view can exclude sensitive columns but include all other columns. Authorizing users and programmers access to the view but not the underlying table protects the sensitive data.
- **Save complex queries**. Complex or difficult SELECT statements can be saved as a view. Database users can reference the view without writing the SELECT statement.
- **Save optimized queries**. Often, the same result table can be generated with equivalent SELECT statements. Although the results of equivalent statements are the same, performance may vary. To ensure fast execution, the optimal statement can be saved as a view and distributed to database users.

For the above reasons, views are supported in all relational databases and are frequently created by database administrators. Database users need not be aware of the difference between view and base tables.

TA, Jordan (A – L).

TA, Fernando (M – Z).

Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(2.14.2024 Attendance X missing LastName.docx)

Example Schemas

- We will use these schemas of the **Sailors**, **Boats**, and **Reserves Relations** in our examples.

Sailors(sid:integer,sname:string,rating:string,age:real)

Boats(bid:integer,bname:string,color:string)

Reserves(sid:integer,bid:integer,day:date)

Example Instances

We will use these instances of the **Sailors**, **Reserves** and **Boats** **Relations** in c
examples.

Sailors

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Boats

B1

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Basic SQL Query

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A';
```

```
SELECT    [DISTINCT]
target-list
FROM      Relation-list
WHERE     qualification
```

- ***Relation-list*** A list of **Relation names** (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of **Relations** in ***Relation-list***
- *qualification* Comparisons (attr *op* const or attr1 *op* attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$ combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. **Default is that duplicates are *not* eliminated!**

Conceptual Evaluation Strategy

```
SELECT  S.name, E.cid
FROM    Students S, Enrolled E
WHERE   S.sid=E.sid AND E.grade='A';
```

```
SELECT      [DISTINCT]
target-list
FROM        Relation-list
WHERE       qualification
```

- Semantics of an **SQL** query defined in terms of the following conceptual evaluation strategy:
  - 1. Compute the join of ***Relation-list***. ⋈
  - 2. Discard resulting tuples if they fail *qualifications*. **σ**
  - 3. Delete attributes that are not in *target-list*. **Π**
  - If DISTINCT is specified, eliminate duplicate rows.

- This strategy is probably the least efficient way to compute a **SQL** query! An optimizer will find more efficient strategies to compute *the same answers*.

Conceptual Evaluation – 1. Compute the join of *Relation-list*.

names of sailors who've reserved boat #103 (Q1)

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

```
SELECT  [DISTINCT] target-list
FROM    Relation-list
WHERE   qualification
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day
22	101	10/10/96
58	103	11/12/96

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/ 10/ 9
22	dustin	7	45.0	58	103	11/ 12/ 9
31	lubber	8	55.5	22	101	10/ 10/ 9
31	lubber	8	55.5	58	103	11/ 12/ 9
58	rusty	10	35.0	22	101	10/ 10/ 9
58	rusty	10	35.0	58	103	11/ 12/ 9

Conceptual Evaluation – 2. Discard resulting tuples if they fa

qualifications.

names of sailors who've reserved boat #103 σ (Q1)

```
ELECT  S.sname
ROM    Sailors S, Reserves R
HERE   S.sid=R.sid AND R.bid=10
```

```
SELECT      [DISTINCT]
target-list
FROM        relation-list
WHERE       qualification
```

(sid)	sname	rating	age	(sid)	bid	day
58	rusty	10	35.0	58	103	11/ 12/

Conceptual Evaluation – 3. Delete attributes that are not in target-list.

names of sailors who've reserved boat #103 (Q1) π

```
ELECT  S.sname
ROM    Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=10
```

```
SELECT      [DISTINCT]
target-list
FROM        relation-list
WHERE       qualification
```

sname
rusty

A Note on Range Variables

- Really needed **only if** the same **Relation** appears twice in the **FROM** clause. This **SQL** query can also be written as:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

OR

It is good style, however, to use Range Variables always!

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid AND bid=103
```

(Q1) Find names of sailors who've reserved boat #103

Reserves

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Boats

B1

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

How many **Relations**? **2**

Result?

sname
rusty

Operations and expressions

In his original paper on the relational model, E. F. Codd introduced formal operations for manipulating tables. Codd's operations, called ***relational algebra***, have since been refined and are the theoretical foundation of SQL. Relational algebra is primarily of theoretical interest but does have practical application in compiling SQL queries.

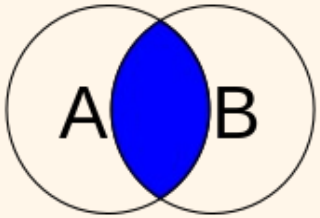
Relational algebra has nine operations. Each operation is denoted with a special symbol, often a letter of the Greek alphabet. Operation symbols can be combined with tables in expressions, just as + - × / can be combined with numbers in arithmetic expressions. Each relational algebra expression is equivalent to an SQL query and defines a single result table.

Table 8.3.1: Relational algebra operations.

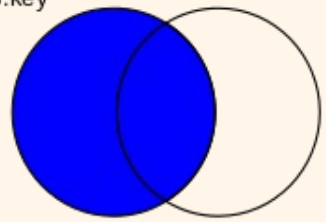
Operation	Symbol	Greek letter	Derivation
Select	σ	sigma	corresponds to Latin letter s, for Select
Project	Π	Pi	corresponds to Latin letter P, for Project
Product	$\times$		multiplication symbol
Join	$\bowtie$		multiplication symbol with vertical bars
Union	$\cup$		set theory
Intersect	$\cap$		set theory
Difference	$-$		set theory
Rename	ρ	rho	corresponds to Latin letter r, for Rename
Aggregate	γ	gamma	corresponds to Latin letter g, for group

SQL JOINS

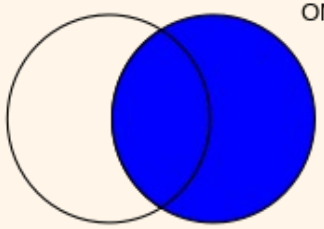
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key



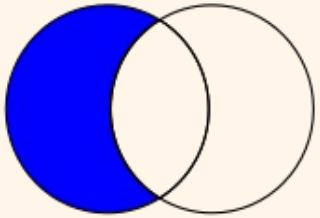
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key



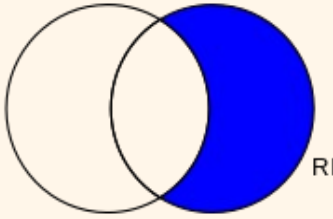
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key



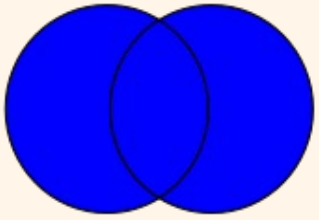
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL



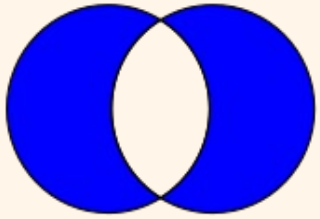
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE a.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key iIS NULL



Suppose you have two tables, with a single column each, and data as follows:

```
A   B
-   -
1   3
2   4
3   5
4   6
```

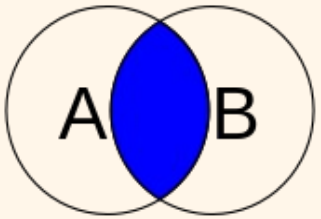
Note that (1,2) are unique to A, (3,4) are common, and (5,6) are unique to B.

Inner join

An inner join using either of the equivalent queries gives the intersection of the two tables, i.e. the **two rows they have in common**.

```
select * from a INNER JOIN b on a.a = b.b;
select a.*, b.*  from a,b where a.a = b.b;
```

```
SELECT <fields>
FROM TableA  A
INNER JOIN TableB  B
ON A.key = B.key
```




```
a | b
--+--
3 | 3
4 | 4
```

Suppose you have two tables, with a single column each, and data as follows:

```
A    B
-    -
1    3
2    4
3    5
4    6
```

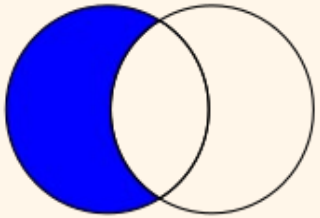
Note that (1,2) are unique to A, (3,4) are common, and (5,6) are unique to B.

Left outer join

A left outer join will give **all rows in A**, plus **any common rows in B**.

```
select * from a LEFT OUTER JOIN b on a.a = b.b;
select a.*, b.*  from a,b where a.a = b.b(+);
```

```
a |  b
--+-----
1 | null
2 | null
3 |    3
4 |    4
```



Suppose you have two tables, with a single column each, and data as follows:

```
A    B
-    -
1    3
2    4
3    5
4    6
```

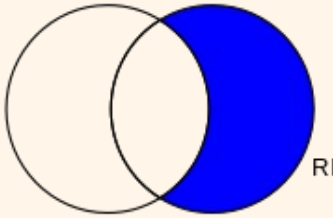
Note that (1,2) are unique to A, (3,4) are common, and (5,6) are unique to B.

Right outer join

A right outer join will give **all rows in B**, plus **any common rows in A**.

select * from a **RIGHT OUTER JOIN b** on a.a = b.b;
select a.*, b.* from a,b where a.a(+) = b.b;

```
a    | b
-----+----
3    | 3
4    | 4
null |  5
null |  6
```



SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE a.key IS NULL

Suppose you have two tables, with a single column each, and data as follows:

```
A    B
-    -
1    3
2    4
3    5
4    6
```

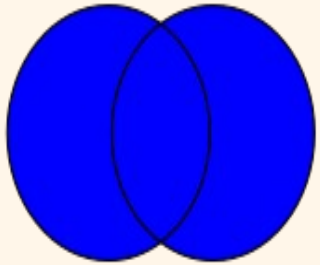
Note that (1,2) are unique to A, (3,4) are common, and (5,6) are unique to B.

Full outer join

A full outer join will give you the **union of A and B**, i.e. all the rows in A and all the rows in B. If something in A doesn't have a corresponding datum in B, then the B portion is null, and vice versa.

select * from a **FULL OUTER JOIN b** on a.a = b.b;

```
 a   |  b
-----+-----
   1 | null
   2 | null
   3 |    3
   4 |    4
null |    6
null |    5
```



Joins

In relational databases, reports are commonly generated from data in multiple tables. Multi-table reports are written with join statements.

A ***join*** is a SELECT statement that combines data from two tables, known as the ***left table*** and ***right table***, into a single result. The tables are combined by comparing columns from the left and right tables, usually with the = operator. The columns must have comparable data types.

Usually, a join compares a foreign key of one table to the primary key of another. However, a join can compare any columns with comparable data types.

PARTICIPATION ACTIVITY | 8.1.1: Example join.

Department

Code	DepartmentName	Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

ID	EmployeeName	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

```
SELECT DepartmentName, EmployeeName
FROM Department, Employee
WHERE Manager = ID;
```

Result

DepartmentName	EmployeeName
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

PARTICIPATION ACTIVITY | 8.1.2: Joins.

1) Which columns can be compared in a join?

- ○ Only primary and foreign key columns.
- ◉ Only columns with comparable data types.
- ○ Any columns.

Correct

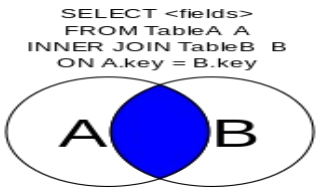
A join can compare any columns with comparable data types. Most often, a foreign key of one join table is compared to the primary key of another.

?????

Inner and outer joins

Matching left and right table rows always appear in a join. In some cases, the database user may also want to see unmatched rows from left, right, or both tables. To enable all cases, relational databases have four types of joins:

- An ***inner join*** selects only matching left and right table rows.
- A ***left join*** selects all left table rows, but only matching right table rows.
- A ***right join*** selects all right table rows, but only matching left table rows.
- A ***full join*** selects all left and right table rows, regardless of match.



An ***outer join*** is any join that selects unmatched rows, including left, right, and full joins. The result table of an outer join may contain unknown values. In the result table of a left join, for example, unmatched left table values are paired with unknown right table values. These unknown values appear as NULL in the result table.

Inner, left, right, and full joins are written with keywords INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN, as in the animations below. The ON clause specifies the join columns. The left table appears in the FROM clause, and the right table appears in the JOIN clause.

PARTICIPATION ACTIVITY 8.1.3: Inner and full joins.

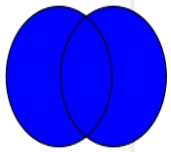
Department

Code	DepartmentName	Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

ID	EmployeeName	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

```
SELECT DepartmentName, EmployeeName
FROM Department      <- left table
INNER JOIN Employee  <- right table
ON Department.Manager = Employee.ID
```



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key

TA time (Fernando) – 3 minutes
(CA 8.8.1 Step 1 – Joins)

CHALLENGE ACTIVITY | 8.8.1: Inner and outer joins.

Join Every row in the left table with every row in the right table

Complete the SQL statement to generate the Result table.

College

•College	○City
Duke	Durham
Stanford	Palo Alto
Brown	Providence
Emory	Atlanta

Location

•City	StateCode
Durham	NC
Palo Alto	CA
Providence	RI
Atlanta	GA

Left table

Right table

16

```
SELECT College, StateCode
FROM College, Location
WHERE College.City = Location.City;
```

Result

College	StateCode
Duke	NC
Stanford	CA
Brown	RI
Emory	GA

1

$\bowtie$ σ π

Feedback?

CHALLENGE ACTIVITY | 8.8.1: Inner and outer joins.

544874.3144414.qx3zqy7

Jump to level 1

Complete the SQL statement to generate the Result table. The column names in the SELECT clause must match the Result table exactly.

College

• CollegeName	City	Enrollment
Bucknell	Lewisburg	3624
Rutgers	Newark	67556
Emory	Atlanta	14769

Location

• City	StateCode
Claremont	CA
Lewisburg	PA
Newark	NJ
Atlanta	GA

```
SELECT CollegeName, StateCode, Enrollment
FROM College, Location
WHERE College.City = Location.City ;
```

Result

CollegeName	StateCode	Enrollment
Bucknell	PA	3624
Rutgers	NJ	67556
Emory	GA	14769

1 2 3

Check Next

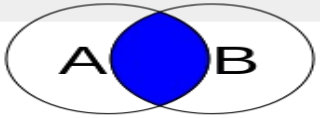
✔ Expected:
CollegeName, StateCode, Enrollment
College.City = Location.City

The column names following SELECT must match the column names in the Result table, in the same order. Since the Result table includes the state code of the location that appears in each college, the WHERE clause must set College.City = Location.City.

TA time (Fernando) – 3 minutes
(CA 8.8.1 Step 2 – Inner Joins)

CHALLENGE ACTIVITY | 8.8.1: Inner and outer joins.

```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```



Check the rows that appear in the result of the SQL statement.

Left table

Order

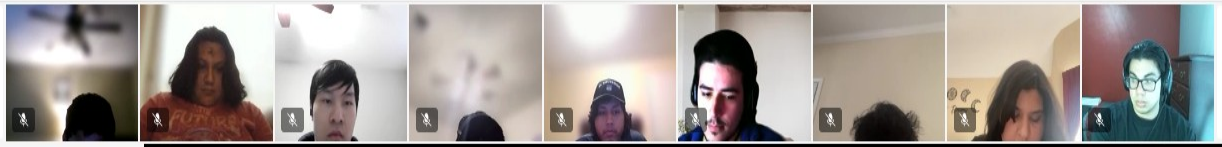
•OrderCode	○PartNumber	Quantity
A36	492	1
R61	827	1
Z23	662	3

Part

• PartNumber	PartName	Price
492	Buzzer	0.40
662	Wingding	11.00
827	Left wizard	1000.00

```
SELECT OrderCode, PartName
FROM Order
INNER JOIN Part
ON Order.PartNumber =
        Part.PartNumber;
```

	OrderCode	PartName
☆ ☐	A36	Buzzer
☐	R61	Buzzer
☐	Z23	Buzzer
☐	A36	Wingding
☐	R61	Wingding
☆ ☐	Z23	Wingding
☐	A36	Left wizard
☆ ☐	R61	Left wizard
☐	Z23	Left wizard
☐	A36	NULL
☐	R61	NULL
☐	Z23	NULL
☐	NULL	Buzzer
☐	NULL	Wingding
☐	NULL	Left wizard



CHALLENGE ACTIVITY 8.8.1: Inner and outer joins.

544874.3144414.qx3zqy7

Jump to level 1

Check the rows that appear in the result of the SQL statement.

Order

OrderCode	PartNumber	Quantity
A36	662	1
R61	662	1
Z23	827	3

Part

PartNumber	PartName	Price
492	Buzzer	0.40
662	Wingding	11.00
827	Left wizard	1000.00

```
SELECT OrderCode, PartName
FROM Order
INNER JOIN Part
ON Order.PartNumber =
        Part.PartNumber;
```

	OrderCode	PartName
☐	A36	Buzzer
☐	R61	Buzzer
☐	Z23	Buzzer
☐	A36	Wingding
☐	R61	Wingding
☐	Z23	Wingding
☐	A36	Left wizard
☐	R61	Left wizard
☐	Z23	Left wizard
☐	A36	NULL
☐	R61	NULL
☐	Z23	NULL
☐	NULL	Buzzer
☐	NULL	Wingding
☐	NULL	Left wizard

1 2 3

Check Next

✔ Expected:
A36, Wingding
R61, Wingding
Z23, Left wizard

The result of an INNER join includes only orders and parts with matching part numbers.

Order A36 is for part number 662 with name Wingding.
Order R61 is for part number 662 with name Wingding.
Order Z23 is for part number 827 with name Left wizard.

No other orders and parts have matching part numbers.

View solution (Instructors only)

Find sailors **Ids** who've reserved at least one boat (Q4)

Reserves

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

```
SELECT  S.sid
FROM  Sailors S, Reserves
WHERE  S.sid=R.sid
```

The join of **Sailors** and **Reserves** (on sid) ensures that for each selected *sid,* the sailor has made a reservation. (If the sailor has not made a reservation, the second step in the conceptual evaluation strategy would eliminate all rows in the **join** that involve that sailor).

Would adding **DISTINCT** to this query make a difference?

NO, sid is the ***key***

What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause? Would adding **DISTINCT** to this variant of the **SQL** query make a difference?

YES, sname is NOT the ***key***

(Q5) Find names of sailors who've reserved a red **or** a green boat.

R1 Reserves

sid	bid	day
22	102	10/10/96
58	103	11/12/96

S1 Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

B1 Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

How many **Relations**? 3

Result?

sname
dustin
rusty

Find sid's of sailors who've reserved a red **or** a green boat (Q5)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day
22	102	10/10/96
58	103	11/12/96

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

UNION: Can be used to compute the union of any two *union-compatible* **sets** of tuples (which are themselves the result of **SQL** queries).

```
SELECT  S.sid
FROM  Sailors S, Boats B,
Reserves R
WHERE  S.sid=R.sid AND
R.bid=B.bid
                AND B.color='red'
              UNION
SELECT  S.sid
FROM  Sailors S, Boats B,
Reserves R
WHERE  S.sid=R.sid AND
R.bid=B.bid
                AND B.color='green'
```

SQL I Practice Questions

8. SET 2 - 1:SQL I

UPDATE, SELECT, and ORDER BY are _____ in an SQL statement.

a. keywords

b. literals

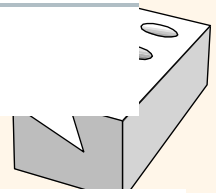
c. identifiers

d. comments

?????

hrke

8. SET 2 - 1:SQL I



A view table provides which benefit when created in a database with multiple tables?

a. A consolidated view of specific data without changing the underlying database structure.

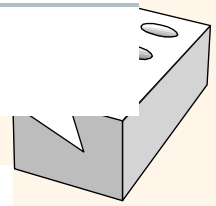
b. A consolidated view of specific data that modifies the underlying source tables.

c. A consolidated view of base table data that is pulled from restructured database tables.

d. A consolidated view of base table data that is pulled from multiple view queries.

?????

hrke



When a database stores view data, it uses a ____ that depends on data in a corresponding _____.

a. base table, view table

b. materialized view, base table

c. view table, materialized view

d. materialized view, view table

?????

hrke

8. SET 2 - 1:SQL I

Which relational algebra expression is equivalent to the following SQL statement?

```
SELECT PassengerName
FROM Booking
WHERE TicketPrice < 1000;
```

a. $\Pi_{(\text{TicketPrice}<1000)}\left(\sigma_{(\text{PassengerName})}(\text{Booking})\right)$

b. $\sigma_{(\text{TicketPrice}<1000)}\left(\Pi_{(\text{PassengerName})}(\text{Booking})\right)$

c. $\Pi_{(\text{PassengerName})}\left(\sigma_{(\text{TicketPrice}<1000)}(\text{Booking})\right)$

d. $\sigma_{(\text{PassengerName})}\left(\Pi_{(\text{TicketPrice}<1000)}(\text{Booking})\right)$

?????

8. SET 2 - 1:SQL I

Refer to the following tables:

Employee

ID	Name	Salary
4428	Amin Shah	42500
5993	Malia Tarkas	53000

Student

ID	Name	Scholarship
1408	Indira Aviz	3000
5993	Malia Tarkas	0
7110	Jiho Chen	4500

How many rows are in the table defined by the following relational algebra expression?

$$\textbf{Employee} \cap \textbf{Student}$$

a. 0

b. 1

c. 2

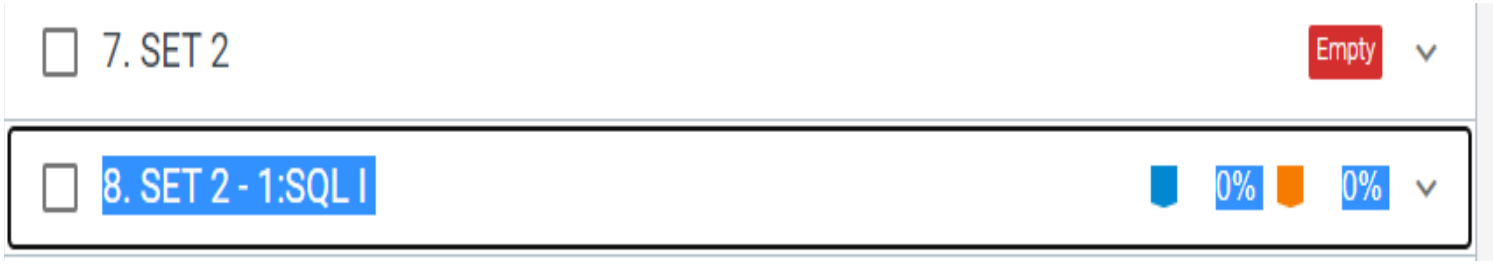
d. 5

?????

hrke

At 5:00 PM .

02.14.2024 (9 – We)	ZyBook SET 2 - 1	Set 2 LECTURE 6 SQL 1



VH work on
SET 2 – 1: SQL I

Next

02.19.2024 (10 - Mo)	ZyBook SET 2 - 2	Set 2 LECTURE 7 SQL II

VH, unHIDE

From 5:05 to 5:15 PM – 5 minutes.

02.14.2024 (9 – We)	ZyBook SET 2 - 1	Set 2 LECTURE 6 SQL 1

CLASS PARTICIPATION 20 points 20% of Total

HAPPY VALENTINE

Class 9 END PARTICIPATION
Not available until Feb 14 at 5:05pm | Due Feb 14 at 5:15pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 9

VH, unhide ZyBook Section 9.

VH, Download Attendance Report
Rename it:
2.14.2024 Attendance Report FINAL

VH, upload Class 9 to CANVAS.