

Databases and MS SQL

Databases are a structured set of data.
MySQL is an open-source relational
database management system.



Outline



1. Intro to Databases
2. ER Model
3. Relationships
4. Normalization
5. Intro to MS SQL
6. Data Creation
7. Transactions
8. Query Data
9. Joins
10. Set Operators
11. Subqueries
12. Procedures
13. Functions
14. Triggers

Prerequisites



1. Microsoft SQL Server:
<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>
2. Windows:
 - MS SQL Server
 - SQL Server Management Studio (IDE)
3. Mac:
 - Docker
 - MS SQL Server
 - Azure Data Studio

Introduction to Databases



What is a Database?

A **database** is a collection of information organized in such a way that a computer program can quickly select desired pieces of data.



Databases

- Data stored on backend database server
- Database types:
 - ◆ *Flat File*
 - ◆ *Hierarchical*
 - ◆ *Network*
 - ◆ *Relational*
 - ◆ *Object Oriented*
 - ◆ *Object Relational*

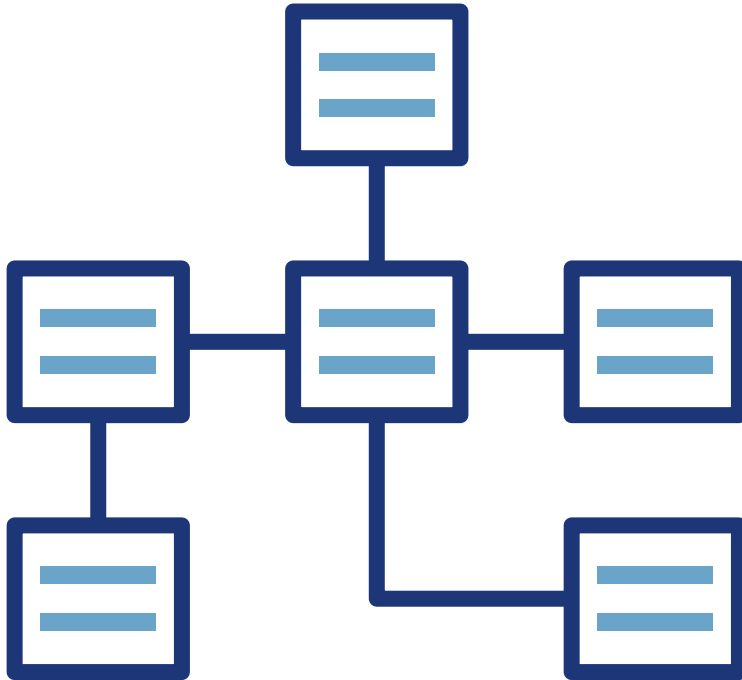


Database Management System (DBMS)

A **DBMS** helps you create and manage databases—like MS Word helps you create and manage word document.



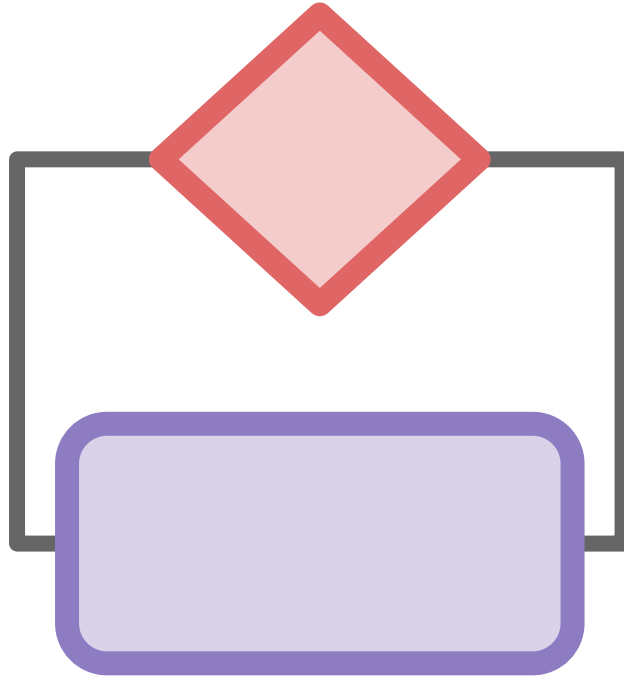
Schemas



Schema - description of the database

Subschema - describes a subset of the database and which users have access to this subset

ER Model



Entity-Relationship (ER) Data Model

An **ER Model** is made up of entities, attributes, and the relationships defined between entities.

Entity

An object in the real world that is distinguishable from other objects.

Ex: Employees, Places

Attribute

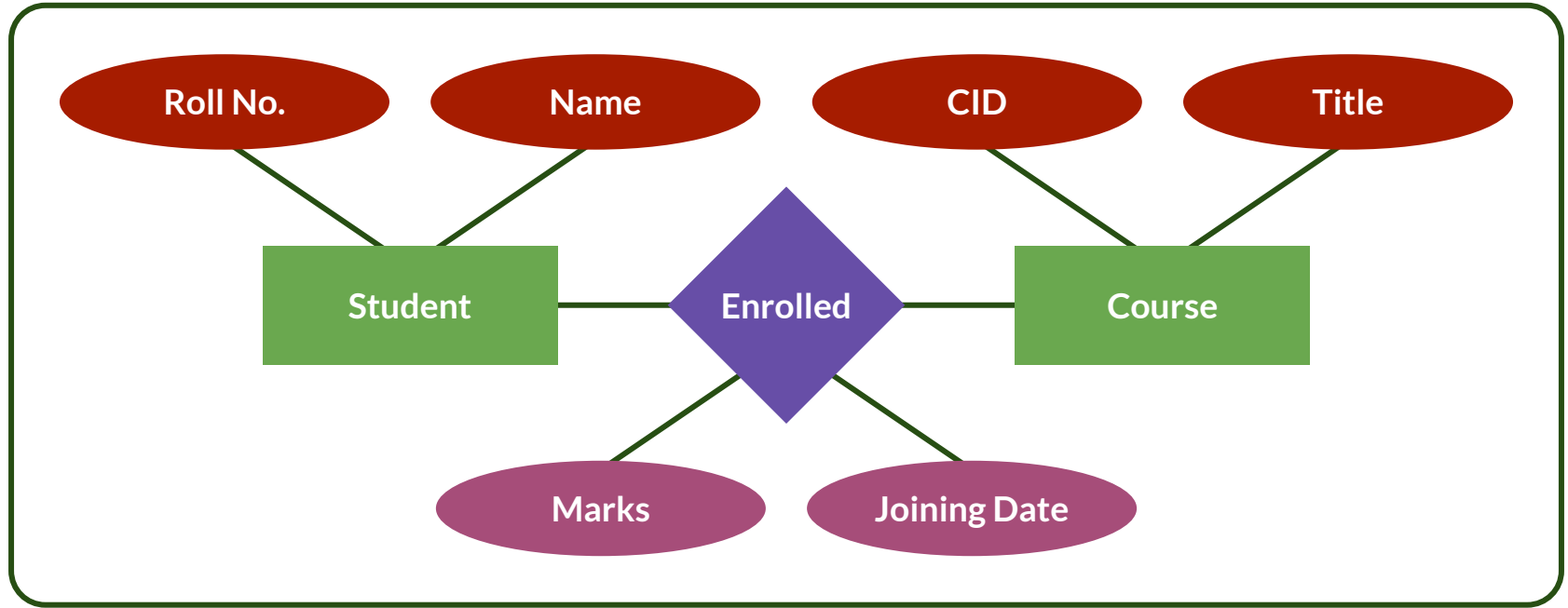
An entity is described in a database based on a set of attributes.

Ex: Name, Age, Gender

Relationship

A relationship links two entities that share one or more attributes.

Ex: Person lives in a House



Entities: Student, Course

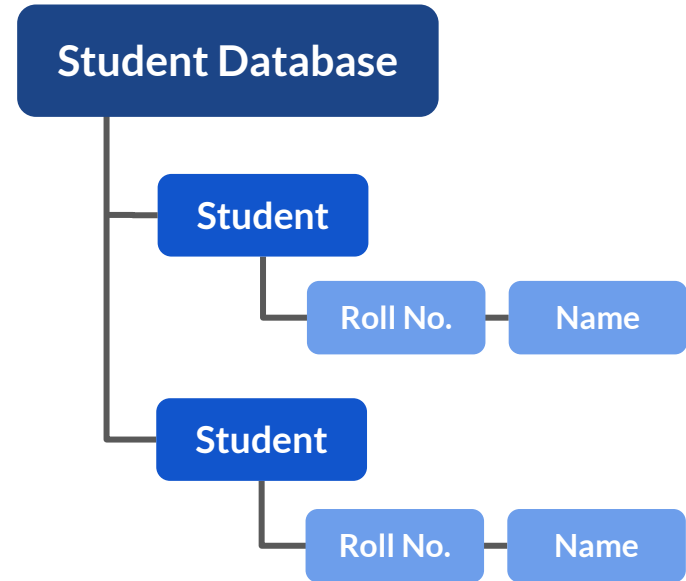
Attributes: Roll No, Name, CID, Title, Marks, Joining Date

Relationship: Enrolled

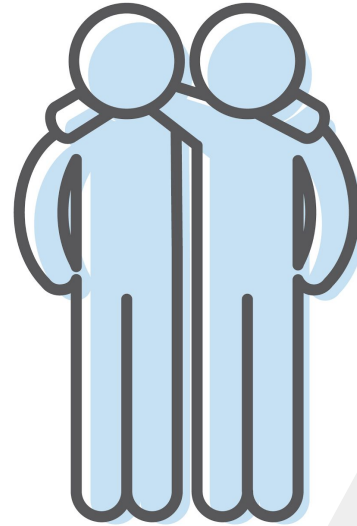
Defining Data With the ER Model

Instance (Record, Tuple) – single, specific occurrence of an entity

- You can have multiple students and each instance will have one student's RollNo and Name

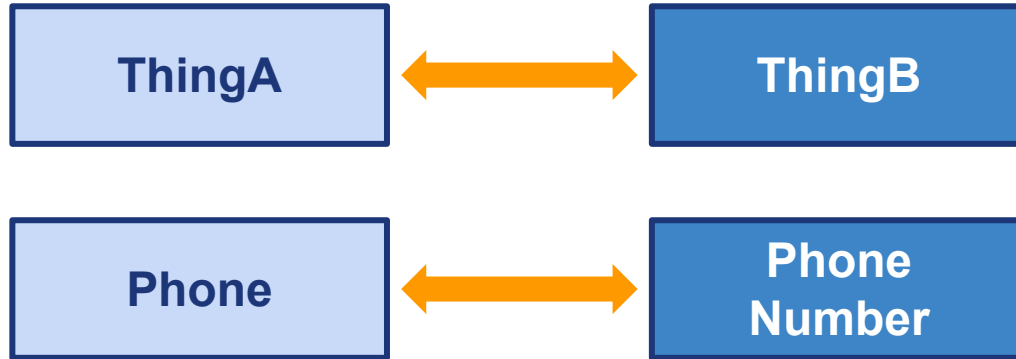


Relationships



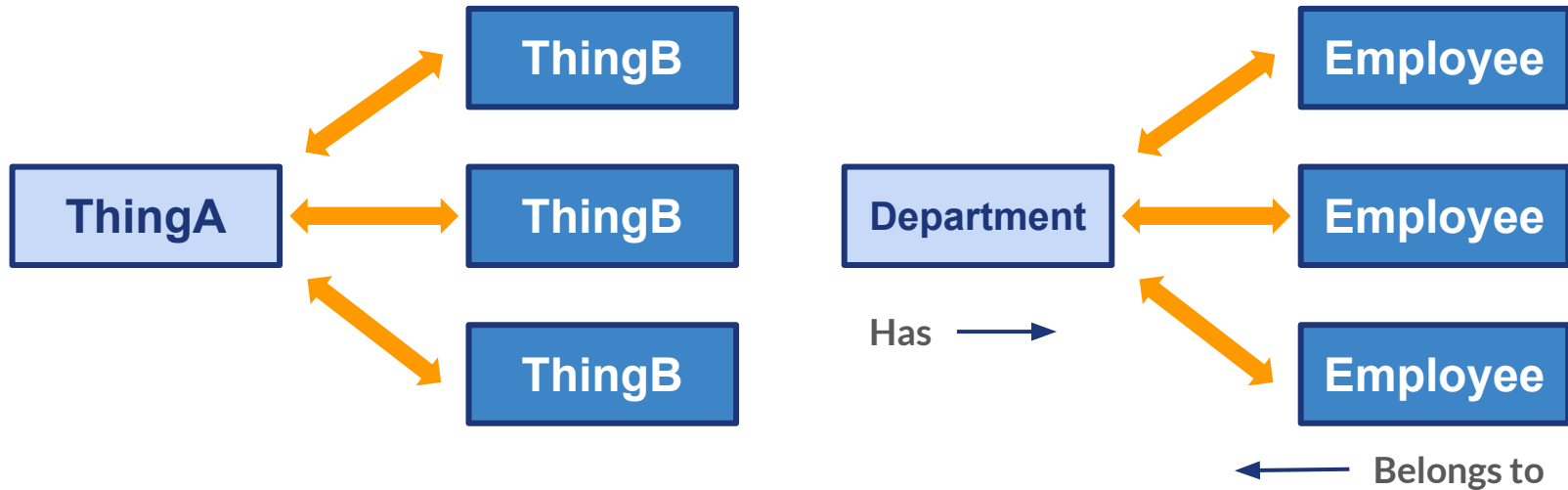
One-to-One

ThingA cannot be related to more than one *ThingB*
ThingB cannot be related to more than one *ThingA*



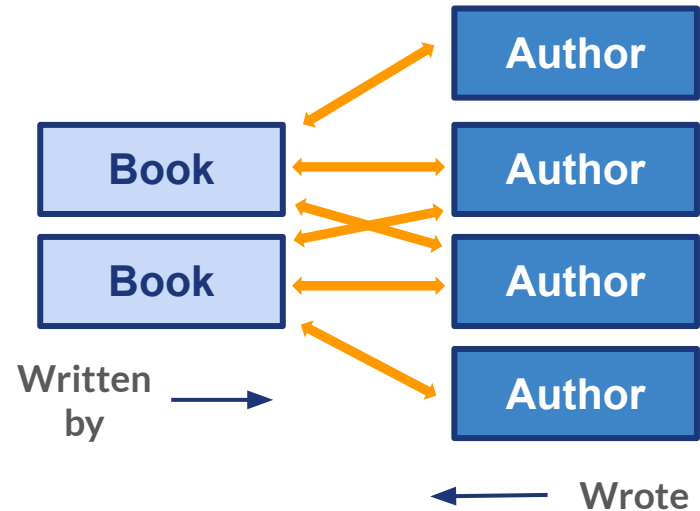
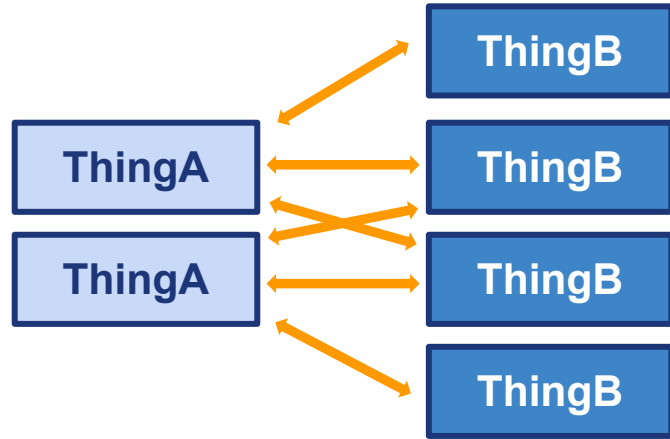
One-to-Many

ThingA can be related to more than one *ThingB*
ThingB cannot be related to more than one *ThingA*



Many-to-Many

ThingA can be related to more than one *ThingB*
ThingB can be related to more than one *ThingA*



Relational Model

- Data is viewed as existing in a two dimensional table known as relations
- A relation (table) consists of unique attributes (columns) and tuples (rows)

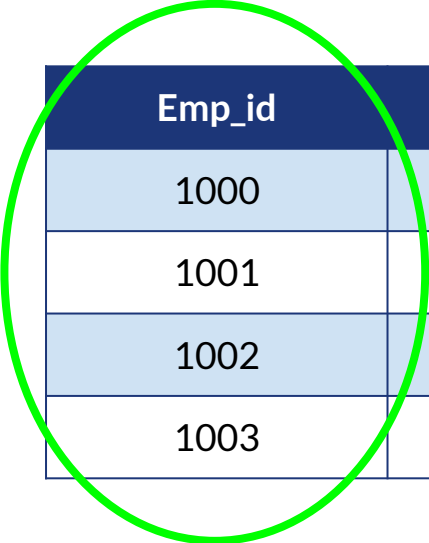
Attributes/Fields/Columns

Emp_id	Emp_name	Emp_age	Emp_email
1000	Derek	24	dk@cognixia.com
1001	Maria	23	ma@cognixia.com
1002	Luis	25	ls@cognixia.com
1003	Janel	25	jl@cognixia.com

Rows/
Records/
Tuples

Keys in Relational Models

Primary Key - uniquely identifies each record in a database table



Emp_id	Emp_name	Emp_age	Emp_email
1000	Derek	24	dk@cognixia.com
1001	Maria	23	ma@cognixia.com
1002	Luis	25	ls@cognixia.com
1003	Janel	25	jl@cognixia.com

Keys in Relational Models

Foreign Key - an attribute in a table that references the primary key of another table

Emp_id	Emp_name	Emp_age	Emp_email	Fk_dept_id
1000	Derek	24	dk@cognixia.com	1
1001	Maria	23	ma@cognixia.com	3
1002	Luis	25	ls@cognixia.com	4
1003	Janel	25	jl@cognixia.com	3

Pk_dept_id	Dept_name
1	Accounting
2	Payroll
3	Marketing
4	HR



REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED



Normalization



Normalization

Normalization – method for organizing data into a database to eliminate data repetition and undesirable characteristics like

- *Insertion anomalies*
- *Delete anomalies*
- *Update anomalies*



Problems Without Normalization

What are the issues we can run into with the table below?

rollNo	name	department	hod	office_tel
1000	Daisy	CS	Mr. X	53337
1001	Nick	CS	Mr. X	53337
1002	Ana	CS	Mr. X	53337
1003	James	CS	Mr. X	53337

Normalization Example: Un-Normalized

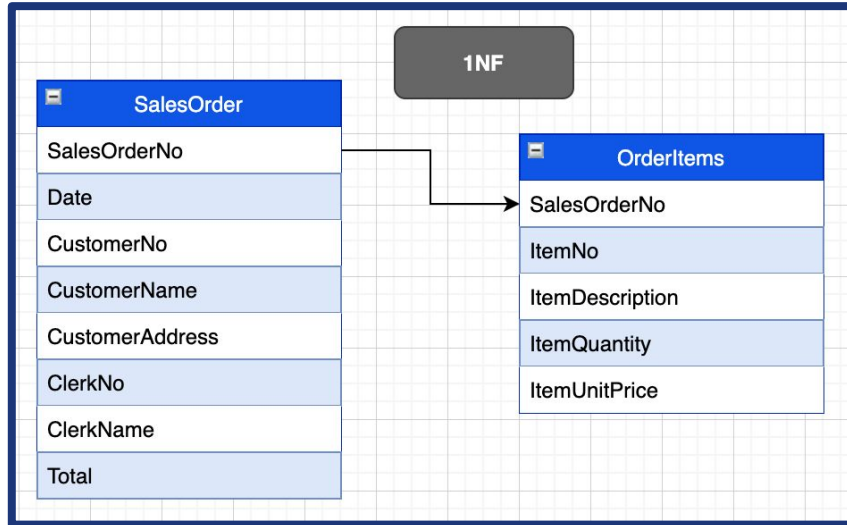
SalesOrders		
SalesOrderNo	ClerkName	Item2UnitPrice
Date	Item1Description	Item3Description
CustomerNo	Item1Quantity	Item3Quantity
CustomerName	Item1UnitPrice	Item3UnitPrice
CustomerAddress	Item2Description	Total
ClerkNo	Item2Quantity	

Normalization Into First Normal Form

- **Separate repeating groups** into new tables
- Start a new table for repeating data
- The **primary key** for the repeating group is usually a **composite key**



Normalization Example: 1NF



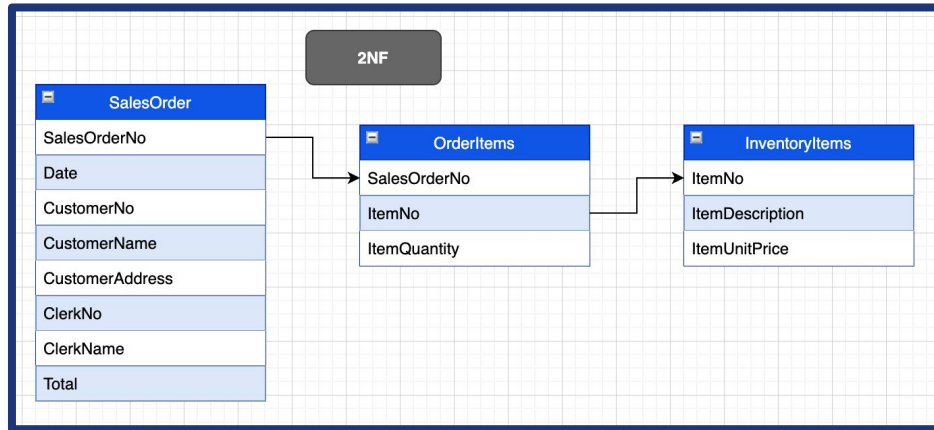
- **OrderItems** table created to separate repeating item information
- Create **ItemNo** and pair it with **SalesOrderNo** as primary key for new table to uniquely identify each item

Normalization Into Second Normal Form

- Remove **partial dependencies** – an attribute depends on only part of the primary key
- Start a new table for partially dependent data and part of key it depends on



Normalization Example: 2NF



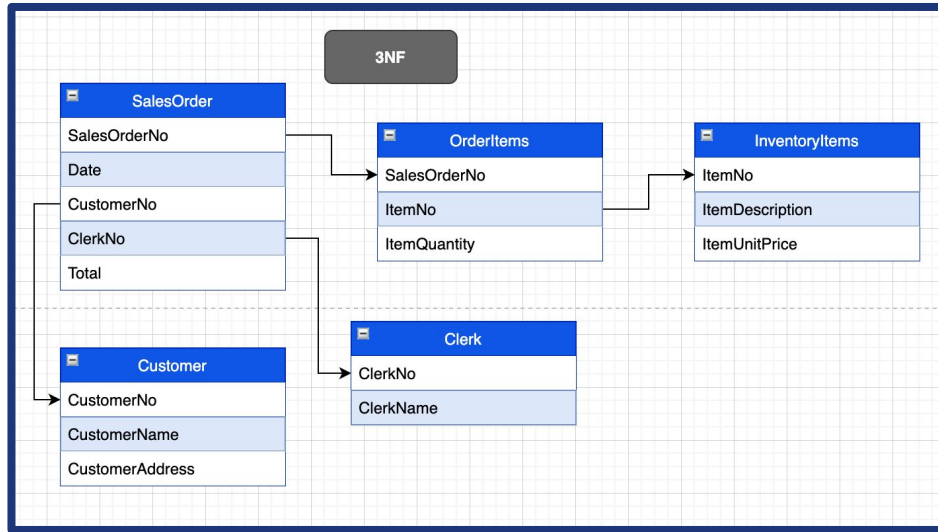
- Removed the partial dependency caused by Item
- Item description not duplicated
- Don't need to insert sales order to add inventory item
- Item information stays even if sales order is deleted
- To change an item description, you don't need to go through each sales order

Normalization Into Third Normal Form

- Remove **transitive dependencies** – attribute depends on an attribute other than the primary key
- Start a new table for transitive dependency and attributes that depend on it
- Keep a copy of the key attribute in the original table



Normalization Example: 3NF



- Removed the transitive dependencies with Customer and Clerk tables
- Now customer and clerk information isn't in every order
- If you delete a sales order, you aren't deleting customer or clerk information
- Don't need to change all orders when you need to update customer or clerk information

Database Planning

- **UML Diagrams** provide a way to visualize and design software or any system
- Flexibly used in development of applications but also to model how data will look
- Can plan out normalized database and determine if relationships between entities will work

Scenario: You are building an application for a music streaming company. They need to store information for artists names, album names, album release date, genre, song name, song length. What kind of UML do you build to model your data?

REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED



Introduction to Microsoft SQL





Microsoft® SQL Server®

Microsoft SQL is a relational database management system (RDBMS) developed and marketed by Microsoft. SQL Server worked exclusively on Windows environment for more than 20 years. In 2016, Microsoft made it available on Linux.

SQL Server Architecture

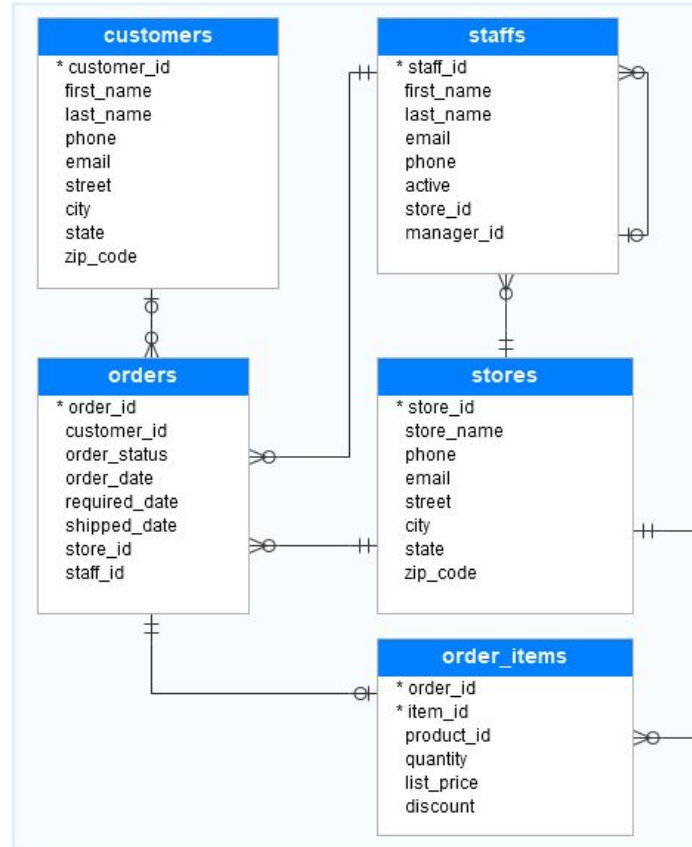
- Consists of two main components:
 - ◆ **Database Engine** - core component of server, consists of *relational engine* that processes queries and a *storage engine* that manages data storage and retrieval.
 - ◆ **SQLOS** - stands for SQL Operating System, provides OS services like memory and I/O management



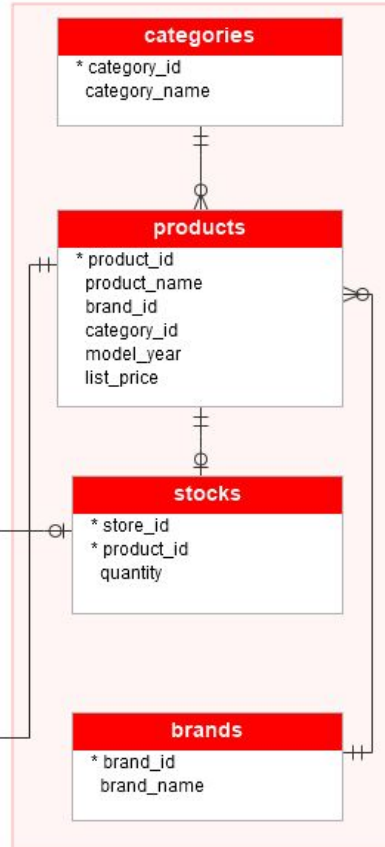
Data Creation



Sales



Production



Before moving forward, for the following examples run, we will be using this sample database called **BikeStores**. It keeps track of sales for a business that sells different types of bikes. Feel free to reference this slide and this [site](#) for information on this database. *Script files will be given to you, please run them now.*

Exact Numeric Types

Table Header	Table Header
<code>bigint</code> , <code>int</code> , <code>smallint</code> , <code>tinyint</code>	Store whole numbers up to 8 bytes, 4 bytes, 2 bytes, and 1 byte in that order.
<code>bit</code>	Stores 0 or 1 or NULL, good for true or false values.
<code>decimal</code> , <code>numeric</code>	Used interchangeably to create decimal values between 5 and 17 bytes.
<code>money</code>	Stores currency values as high as 900 trillion.
<code>smallmoney</code>	Stores smaller currency values as high as 200 thousand.

Exact numeric data types store exact numbers such as integer, decimal, or monetary amount.

Approximate Numeric Types

Table Header	Table Header
<code>float</code>	Floating precision number data from $-1.79E + 308$ to $1.79E + 308$. Can have a number precision of 7 digits. Memory allocation for value will vary based on how much is specified when initialized.
<code>real</code>	Floating precision number data from $-3.40E + 38$ to $3.40E + 38$. Can have a number precision of 15 digits. Only needs 4 bytes of memory allocated per value.

Approximate numeric data types store floating point numeric data and are often used in scientific calculations.

Date & Time Types

Table Header	Table Header
<code>date</code>	Store a date only. From January 1, 0001 to December 31, 9999.
<code>time</code>	Store a time only to an accuracy of 100 nanoseconds.
<code>datetimeoffset</code>	The same as <code>datetime2</code> but with time zone support.
<code>datetime2</code>	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds.

Date & time data types store date and time data. The types above are what should be used for development of newer applications since have larger range and more precise than older types like *`datetime`* and *`smalldatetime`*.

Character Strings Types

Table Header	Table Header
<code>char</code>	Store a fixed character string. Will store strings that have the exact number of characters specified.
<code>varchar</code>	Variable width character string, store up to the amount of characters specified.
<code>text</code>	Like varchar, stores a varied width of characters, but has a max of 2GB of text data it can store.

Character string data types allow you to store fixed length and variable length character data.

Binary String Types

Table Header	Table Header
<code>binary</code>	Fixed width binary string.
<code>varbinary</code>	Variable width binary string.
<code>image</code>	Variable width binary string (stores up to 2GB). Used specifically for storing images using binary.

Binary string data types allow you to store binary data. Usually this binary data is used when dealing with media like images, pdfs, or any data that can not be stored easily with numeric or character data types.

Constraints



Constraints define rules that allow/restrict values stored in columns

- **NOT NULL** - none of the values in this column can be null
- **UNIQUE** - each value in a column or group of columns must be unique, can accept one null value
- **PRIMARY KEY** - set column as primary key, no nulls accepted, must be unique
- **FOREIGN KEY** - set column as foreign key
- **DEFAULT** - if no value given, set default value for column
- **CHECK** - does a boolean check if values in that column meet condition given

Data Definition Language

- **Data Definition Language (DDL)** - define database structures and schemas
- *Doesn't define actual data*, but the things that will hold/contain the data
- Commands:
 - ◆ **CREATE**
 - ◆ **ALTER**
 - ◆ **DROP**
 - ◆ **TRUNCATE**



Data Manipulation Language

- **Data Manipulation Language (DML)** is used to manage data within schema objects
- Manipulate actual data inside tables by updating, adding, or removing
- Commands:
 - ◆ **INSERT**
 - ◆ **UPDATE**
 - ◆ **DELETE**
 - ◆ **SELECT**



Transactions



First Transaction

```
INSERT INTO chef(chef_id, name, best_dish, rest_id)
VALUES(null, 'Chef Ramsey', 'Lamb', 1);

INSERT INTO chef(chef_id, name, best_dish, rest_id)
VALUES(null, 'Chef Remy', 'Ratatouille', 1);

INSERT INTO chef(chef_id, name, best_dish, rest_id)
VALUES(null, 'Swedish Chef', 'Swedish Fööd', 2);
```

*** COMMIT TRANSACTION ***

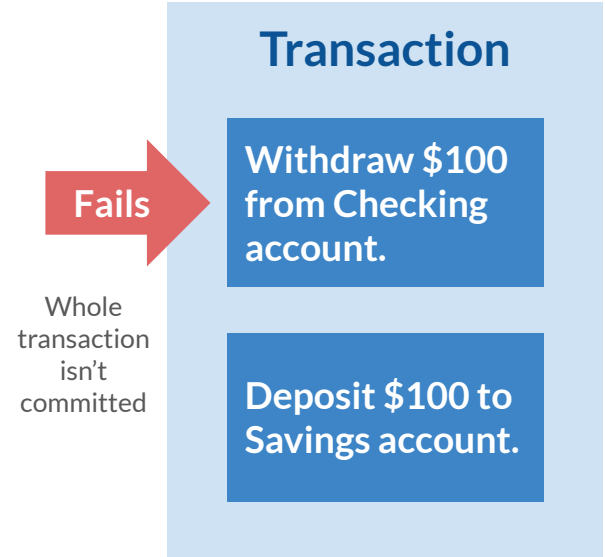
Second Transaction

```
UPDATE chef SET best_dish = 'pizza' WHERE rest_id = 1;
DELETE FROM chef WHERE best_dish = 'steak';
```

*** COMMIT TRANSACTION ***

ACID Properties

1. **Atomicity** - All statements in a transaction go through or none of them happen.
2. **Consistency** - Transactions are permanent once committed. If transaction fails, database remains in state before transaction attempted.
3. **Isolation** - Each transaction is independent and won't interfere with another transaction.
4. **Durability** - System failures or restarts don't affect committed transactions, committed transactions are never "lost".



Transaction - Made up of one or more SQL statements (DML ones). Can be committed manually to database or commit whenever a DDL or DCL command is run.

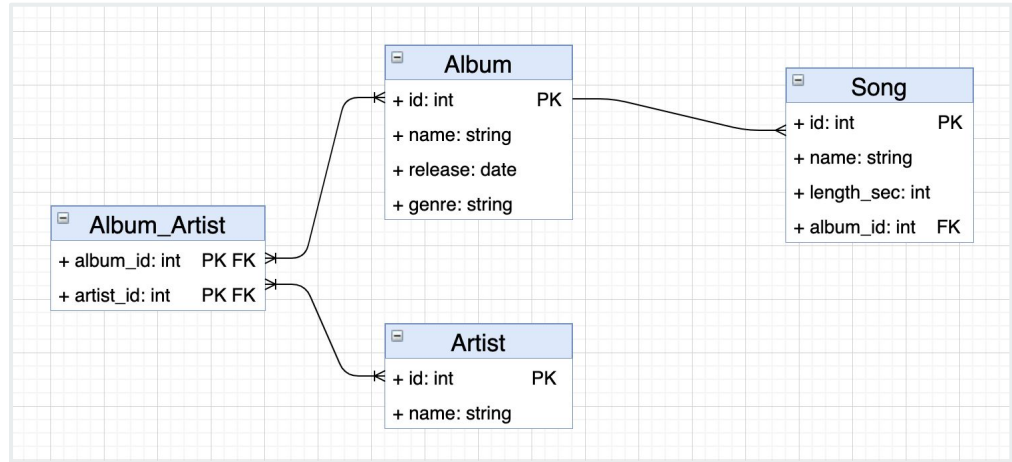
REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED



Database Script Exercise

Take the UML diagram shown below from the previous exercise. Create a database that models this UML by creating a script for a database called **music**, with the tables for **song**, **album**, and **artist**, along with inserting *a few pieces of test data*.

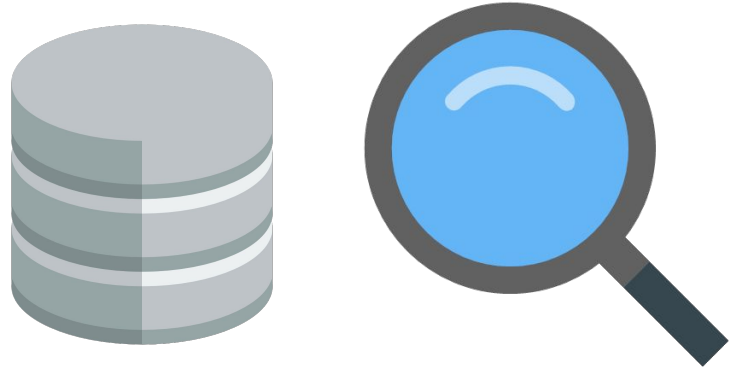


Query Data



SELECT

- Most times, databases are created and designed by someone else
- You will interact and pull data using simple and complex queries using **SELECT**
- To start need to know about:
 - ◆ *Column Aliases*
 - ◆ *WHERE*
 - ◆ *ORDER BY*
 - ◆ *DISTINCT*
 - ◆ *Logical Conditions*
 - ◆ *Comparison Conditions*
 - ◆ *OFFSET*
 - ◆ *FETCH*



Displaying Data in Queries

- **Column Aliases** - give a column a different name when being displayed in a selection
- **WHERE** - used to attach a condition that will limit rows returned by query
- **ORDER BY** - returns data in ascending or descending order
- **DISTINCT** - returns unique/distinct values for columns specified
- **Logical Conditions** - using AND, OR, and NOT to specify multiple conditions and negate conditions
- **Comparison Conditions** - used to compare numbers against groups of numbers and ranges
- **OFFSET** - skip rows of data in a query, an option for ORDER BY clause
- **FETCH** - limit number of rows returned in a query, an option for ORDER BY clause
- **TOP** - limits number of rows returned by a query, stand alone

Grouping Data

- Along with analyzing and filtering data based on each individual row, data can be grouped and be evaluated as a group
- **GROUP BY** is a clause that groups rows in a table and allows these groups to be selected
- Can be used in combination with:
 - ◆ **HAVING** - like **WHERE**, can specify conditions for groups of data instead of rows
 - ◆ **Aggregate Functions** - perform calculations on a set of values (in a column)
 - ◆ **OVER & PARTITION BY** - allow you to select rows and groups together in one results table

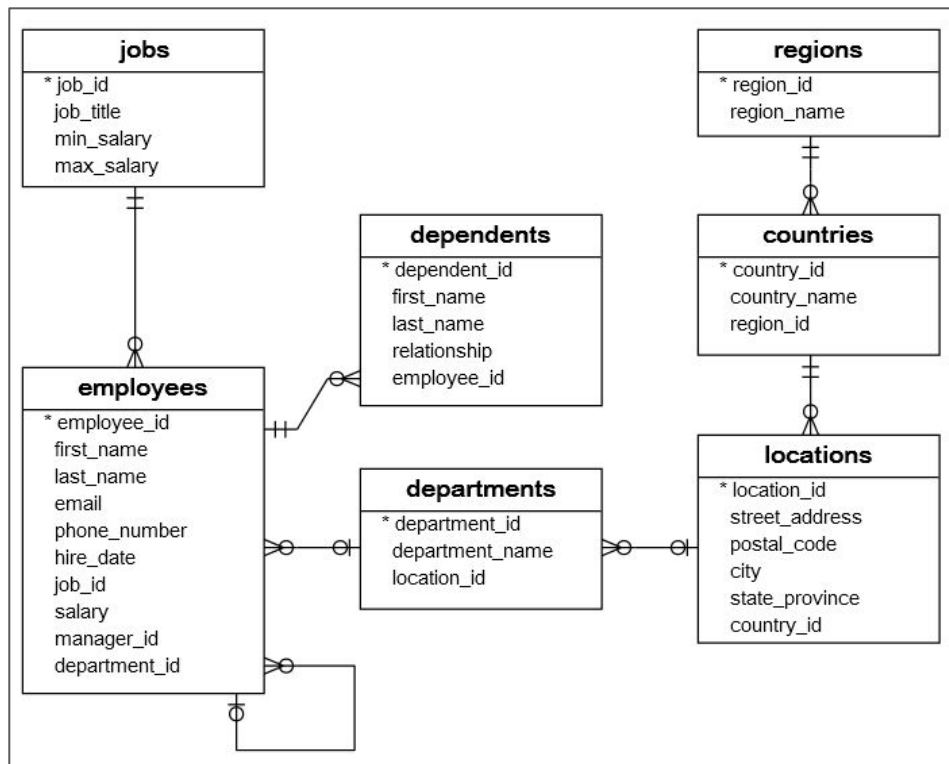
REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED



Query Exercise: HR

You will receive a database script for a Human Resources department called HR_DB. It's structure will look like the following:



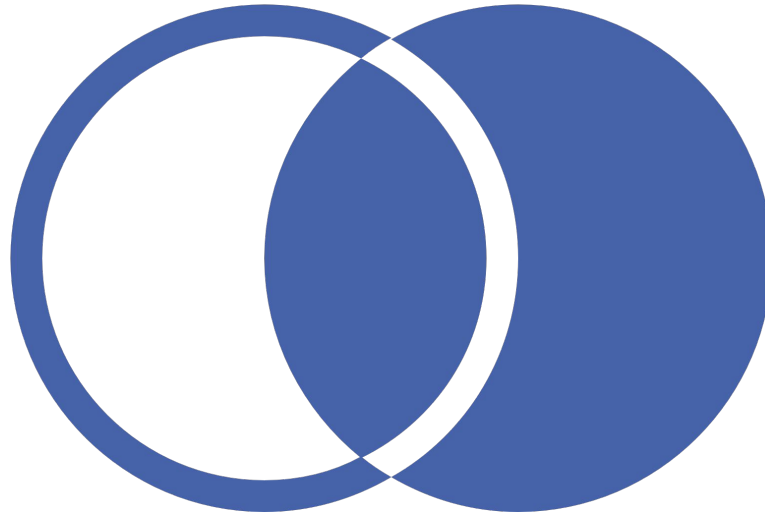
Query Exercise: HR

<code>employees</code>	Stores the data of employees in company.
<code>jobs</code>	Stores job data for employees, including job title and salary range.
<code>departments</code>	Stores department data for employees.
<code>locations</code>	Stores location of the departments of the company.
<code>countries</code>	Stores data of countries where company does business.
<code>regions</code>	Stores data of regions like Asia or Europe. Countries grouped into these regions.

Query Exercise: HR

1. Find the 3 employees with the lowest salaries
2. Find the employee with the 2nd highest salary who has a phone number that begins with 515.
3. Find the name, salary, and hire date of all employees hired before 1995 and a salary over \$10,000.
4. Get all the unique salaries in the employee table.
5. List each unique salary and how many people make that salary.
6. Group employees by department id, find the average salary of each department. Order them from lowest salary to highest.
7. Find all the employees hired between 1995 - 1997.
 - a. Order the employees by their department id and hire date
 - b. Using OVER & PARTITION BY, list the average salary for their department next to each row
8. Select the average salary of an employee per department, if it is odd when rounded, don't include it

Joins



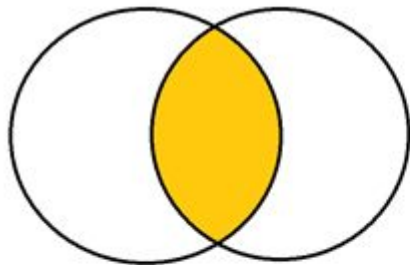
Joins

- A **join** is when you are able to select data from two or more tables as long as those tables have a common field between them (often a foreign key)
- *Can join two tables, multiple tables, or even join a table to itself*
- When doing these joins, can further restrict data by adding conditions with **WHERE** clauses
- Make sure to give **table alias** (other name to refer to the table) in order to set up these conditions and access columns from these tables
 - ◆ Without aliases, difficult to access columns if you need to state the table through the schema

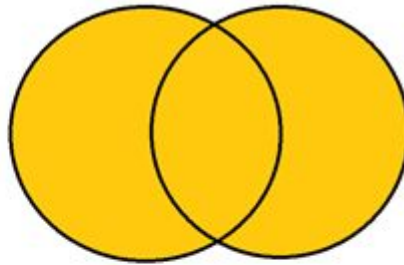
Types of Joins

- **Inner Join** - data that doesn't have an associated common field between tables is not shown
- **Left Join** - data from the right table won't be joined to the left table if the common field on the left table isn't present in the right table
- **Right Join** - same as above, but switch left and right table
- **Full Outer Join** - all rows displayed from both tables regardless of if there is a common field between them

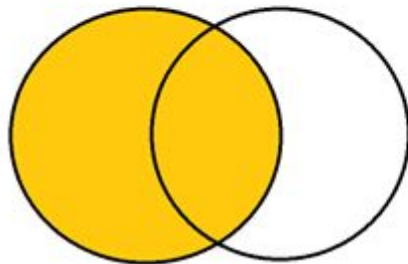
Inner Join



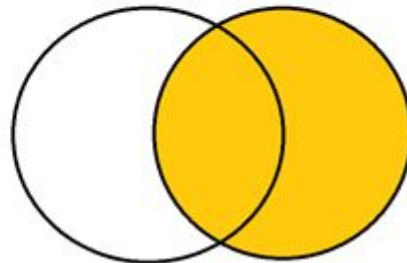
Full Outer Join



Left Outer Join



Right Outer Join

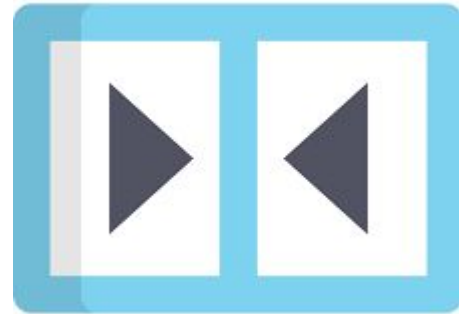


Set Operators

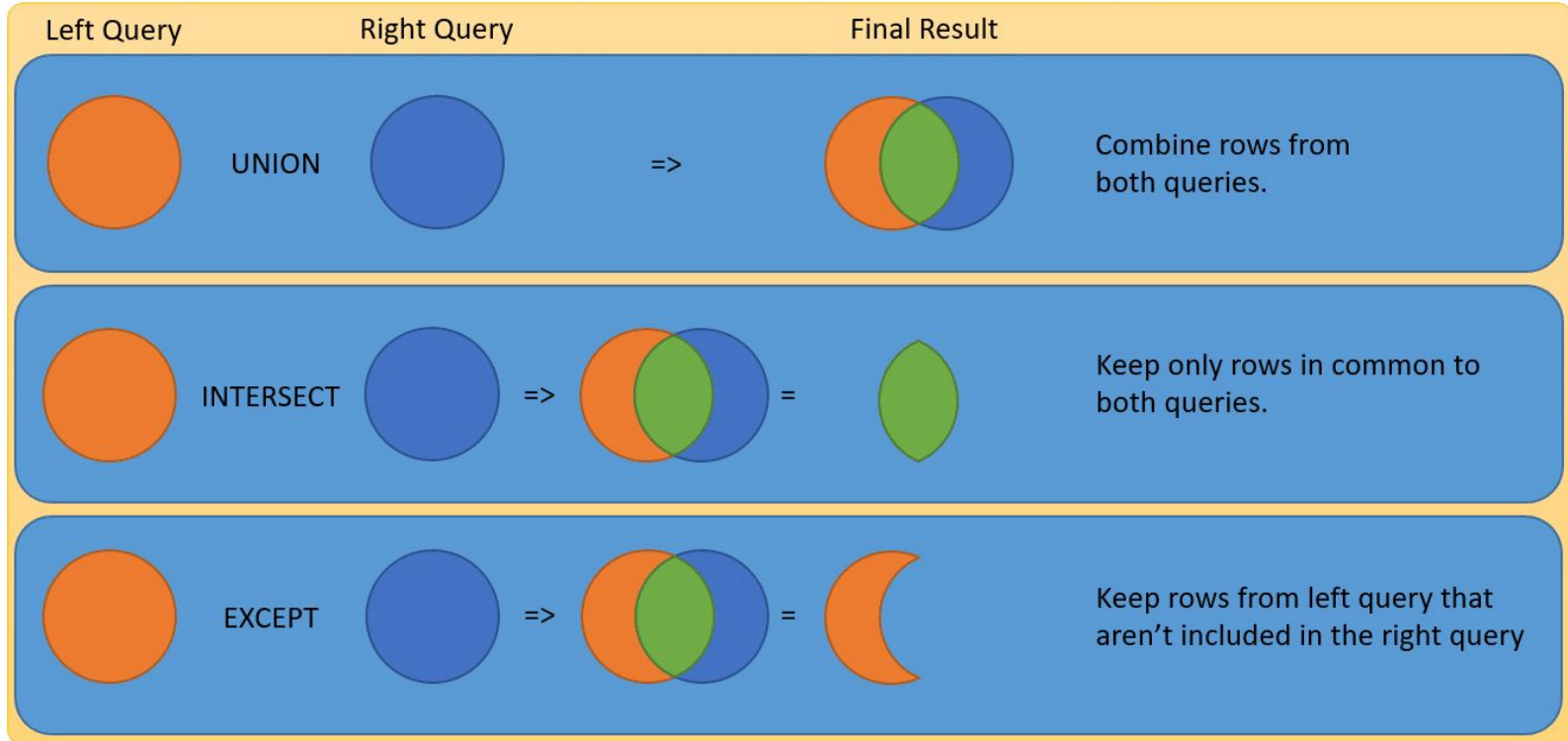


Set Operators

- **Set Operators** are used to select data from two or more queries
- Like joins, are combining data from more than one table, but don't necessarily join the data based on a common field
- Main set operators in MS SQL:
 - ◆ **UNION**
 - ◆ **INTERSECT**
 - ◆ **EXCEPT**



Visual Explanation of UNION, INTERSECT, and EXCEPT operators



REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED

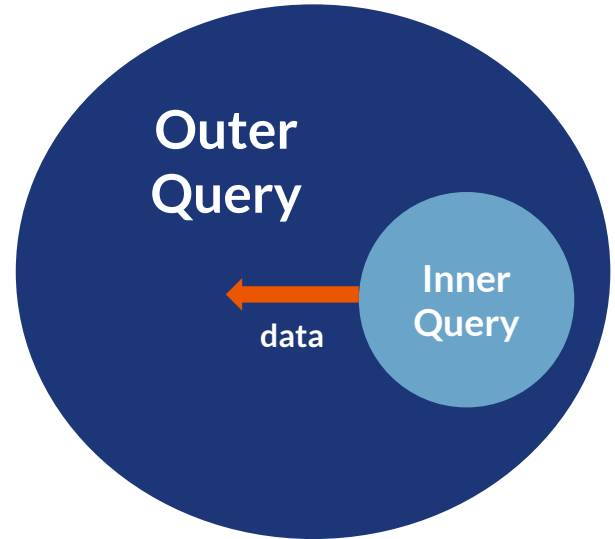


Subqueries



Subquery

- **Subqueries** are queries that are within another query
 - ◆ The **inner query** will return some sort of result or final value to the **outer query**
 - ◆ Outer query then used this result to perform its query
- Can create 32 levels of queries in MS SQL
- Best used when you need to do calculations/evaluations of data and pass those along to another query



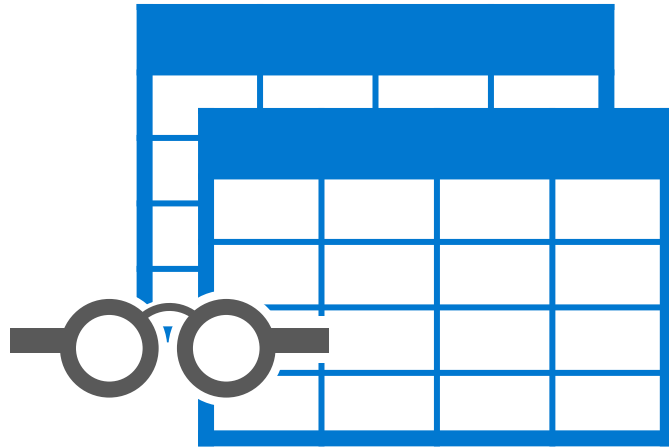
Correlated Subquery

- In regular subqueries, inner queries are unaware of the outer query
- Don't know about the outer query and what it will do
- Sometimes in order to perform certain selections, an *inner query will be aware of the outer query* and use it to do its selection, these are called **Correlated Subqueries**
- In order to be used properly, will need to make sure to set up table aliases to access information from the outer queries

Joins, Set Operations, & Subqueries Exercise

1. Select the first name, last name, job title, and department name for every employee using a join.
2. Select the first name, last name, and email of all the employees who have a dependant using a join and then using a subquery.
3. Create a query that selects the city/province and state from the locations table and the number of employees in each of those locations ordered from highest to lowest.
4. Create a query to get the first and last names of all employees and dependents using UNION.
5. Using EXCEPT select employees who make above the average salary for all employees and remove any employees who are managers.
6. Get the job title, the amount of people with that job title, and the average salary for that job title.

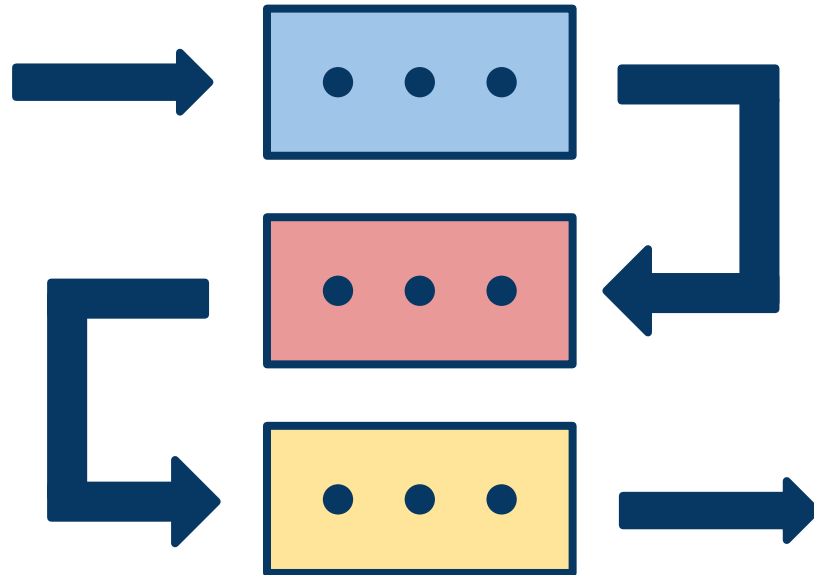
Views



View

- **Views** display virtual tables
- Their contents are based on a base table
- Can think of them as a way to *save a complex select statement*, that way you can call it often without rewriting it each time
- Benefits:
 - ◆ **Security** → can give users access to views instead of full tables
 - ◆ **Simplicity** → can be simple way to view data especially if query you are looking it made up of many conditions and joins
 - ◆ **Consistency** → can be sure the view you access will always be the same without forgetting a long complex query

Procedures



Procedures

- **Stored Procedures** are functions that can save and run multiple SQL statements
- Used when you need to run a set of statements and operations often
- Can take multiple parameters, but can't return values directly
- Think of them like void functions if you are familiar with any programming language



Working With Procedures

- Procedures can take in *parameters* to do their operations
- While they can't return values directly, can use *output parameters* to pass back values they run
- Within these procedures, can work freely to get desired operations by declaring variables and using loops and if statements
- When creating procedures, just like tables and any other structure, they are saved under a schema

Cursor

- **Cursors** are used within procedures to process a result set one row at a time
- Meaning, cursors can go through a query and process/perform some action on each row individually
- Are objects that point to specific rows in a table
- Rarely used queries row by row as often you can use other ways to do so
- Mostly used to access rows one by one to do things like backup data



Functions

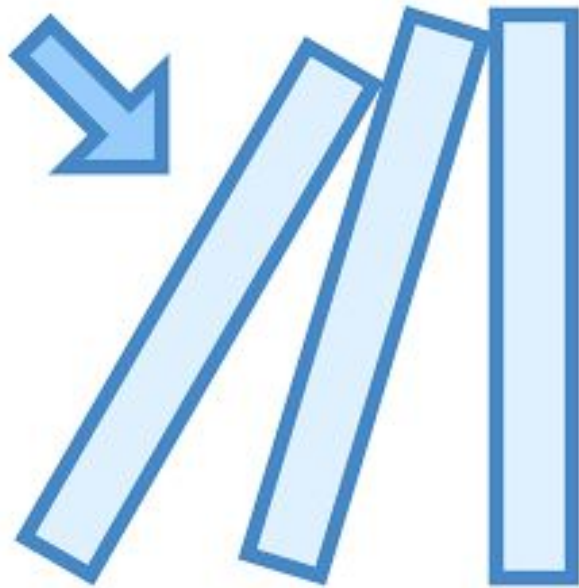


User Defined Functions

- **User Defined Functions** are functions that perform some set of sql statements and return a value
- Big difference between procedure, *procedures don't return values directly*
- Two types:
 - ◆ **Scalar** - return a single value
 - ◆ **Table-Valued** - return a table



Triggers



Triggers

- **Triggers** are a special type of stored procedure that are executed automatically in response to some sort of event
- Three types:
 - ◆ **DML Triggers** - run after a DML statement (INSERT, UPDATE, DELETE) performed
 - ◆ **DDL Triggers** - run after a DDL (CREATE, ALTER, DROP) performed or some system stored procedures that perform DDL-like operations executed
 - ◆ **Logon Triggers** - run after some LOGON events (users login to database)

REVIEW!

1. ANSWER SOME INTERVIEW QUESTION STYLE QUESTIONS
2. BE READY TO ANSWER WHEN YOUR NAME IS CALLED



FIN