Introduction to Databases and SQL

Office of Internal Audit

Ivan Viamontes | July 2020

UNIVERSITY of SOUTH FLORIDA

Introduction

- **Databases** represent a collection of data.
  - Databases are comprised of a number of tables. Within each table, there are rows (also known as a record or tuple) and columns (also known as an attribute).
  - Concept is similar to an Excel spreadsheet. Each sheet is a table. A row is a row within the sheet and the columns are fields.

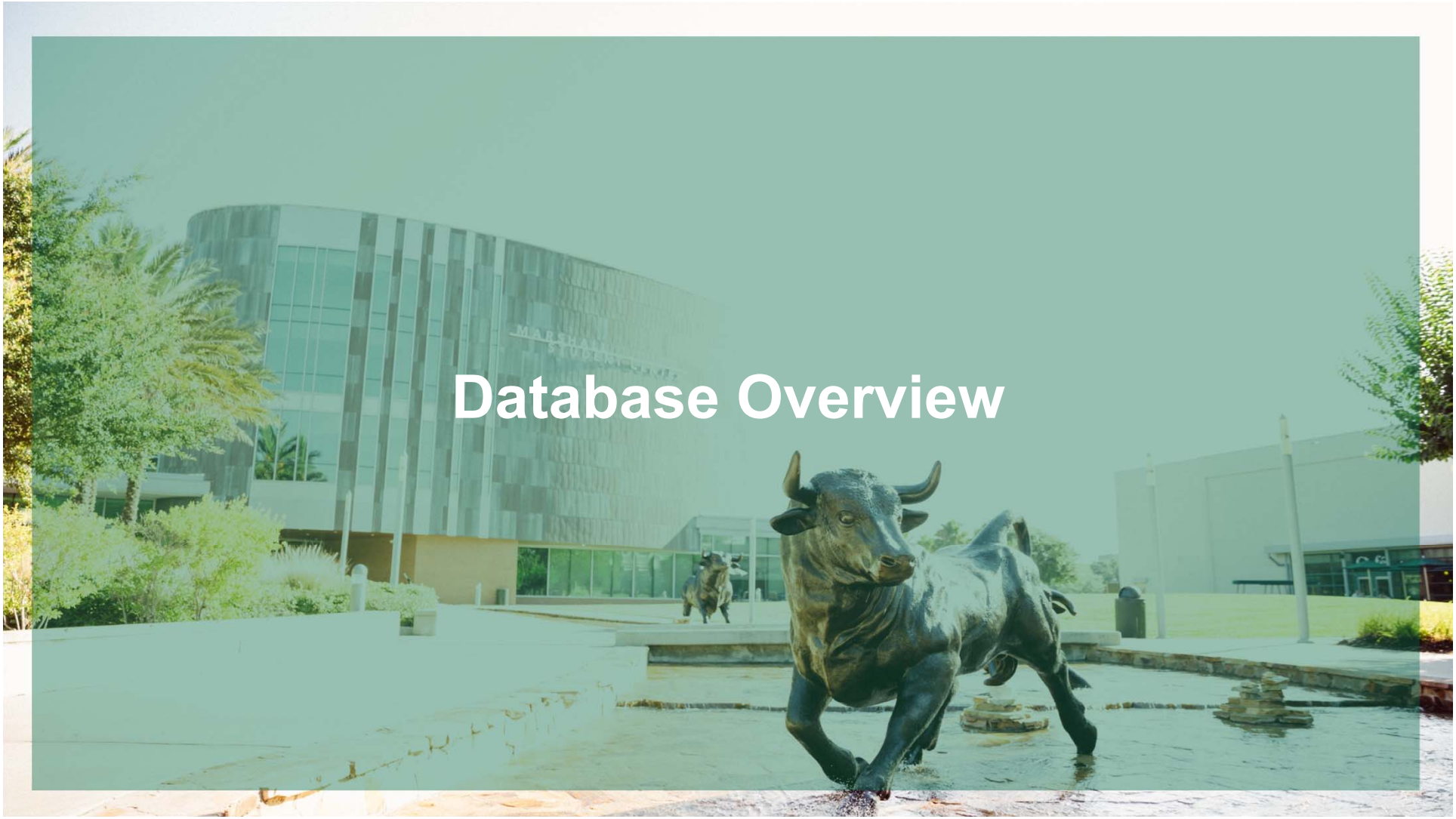
- **Structured Query Language (SQL)** is used to interact with a database. The language was originally developed by IBM in the 1970s.

- The language is intended to mimic the human language and standardize the way a user will interact with a database.

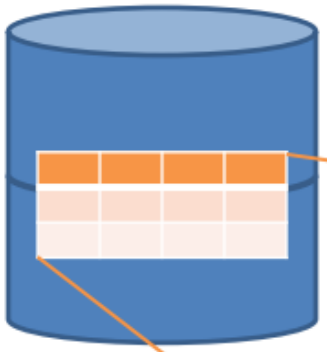
Why Relevant for Business Audits?

Use of Databases within Audits

- All applications rely on databases to store data.
- If there is an error in the code, an incorrect report will be created which can have various impacts to the University.
- Errors can be hard to detect since the business will need to identify all possible scenarios to test the system if they do not understand the code.
- As processes continue to evolve, there is an increased reliance on systems and data. As a result, it is important for a business auditor to understand certain technical components.


Database Overview

Database Example



column

table

row

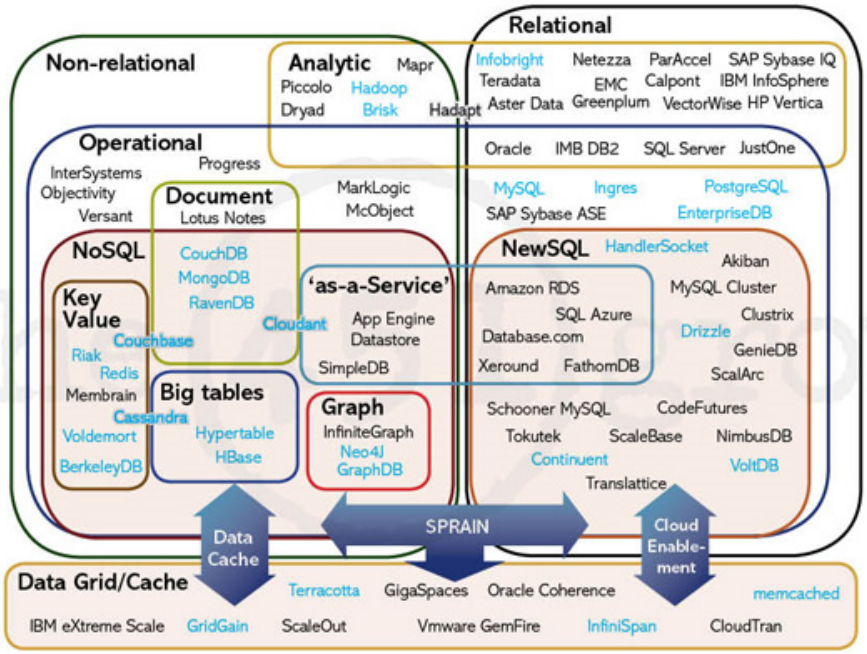
name	email	created_at	updated_at
Shawn	shawn@spbd.gov	2012-12-10 T 10:00 UTC	2013-02-16 T 14:00 UTC
Gus	gus@sbpd.gov	2012-12-01 T 07:00 UTC	2013-02-17 T 09:00 UTC

Types of Databases

- **Relational Databases (RDBMS)**
  - This model organizes data into one or more tables (or "relations") of columns and rows, with a unique key identifying each row.
  - Each table/relation represents one "entity type" (such as student or class schedule).
  - Most common type you will see at the University.

- **Non-Relational Databases (such as NoSQL databases)**
  - Also commonly referred to as Big Data
  - Used in cases where a large amount of data needs to be retrieved and maintained (such as Facebook, Twitter, or Amazon).
  - Despite it’s name, SQL can be used on these databases.

Database Vendors by Type



Source: https://www.infoq.com/news/2011/04/newsql/

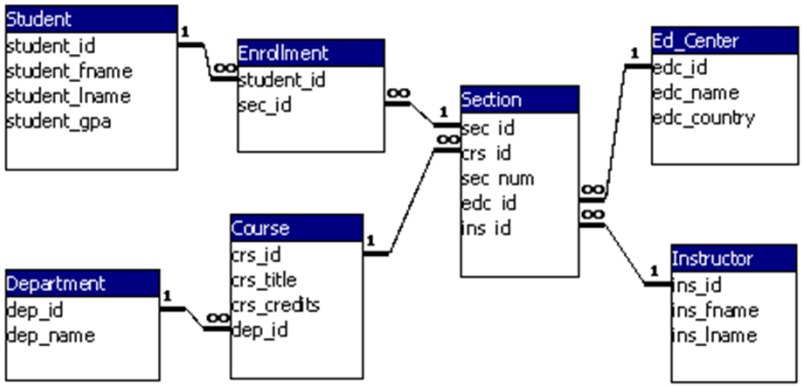
Database Schema

- Databases are organized based on the application being supported.

- A **Database Administrator (DBA)** will often employ a technique known as **database modeling** to create the **database schema** (the actual implementation of the database).

- The database model will describe the tables (e.g., employee, course, buildings, etc…) and the relationships to other tables.

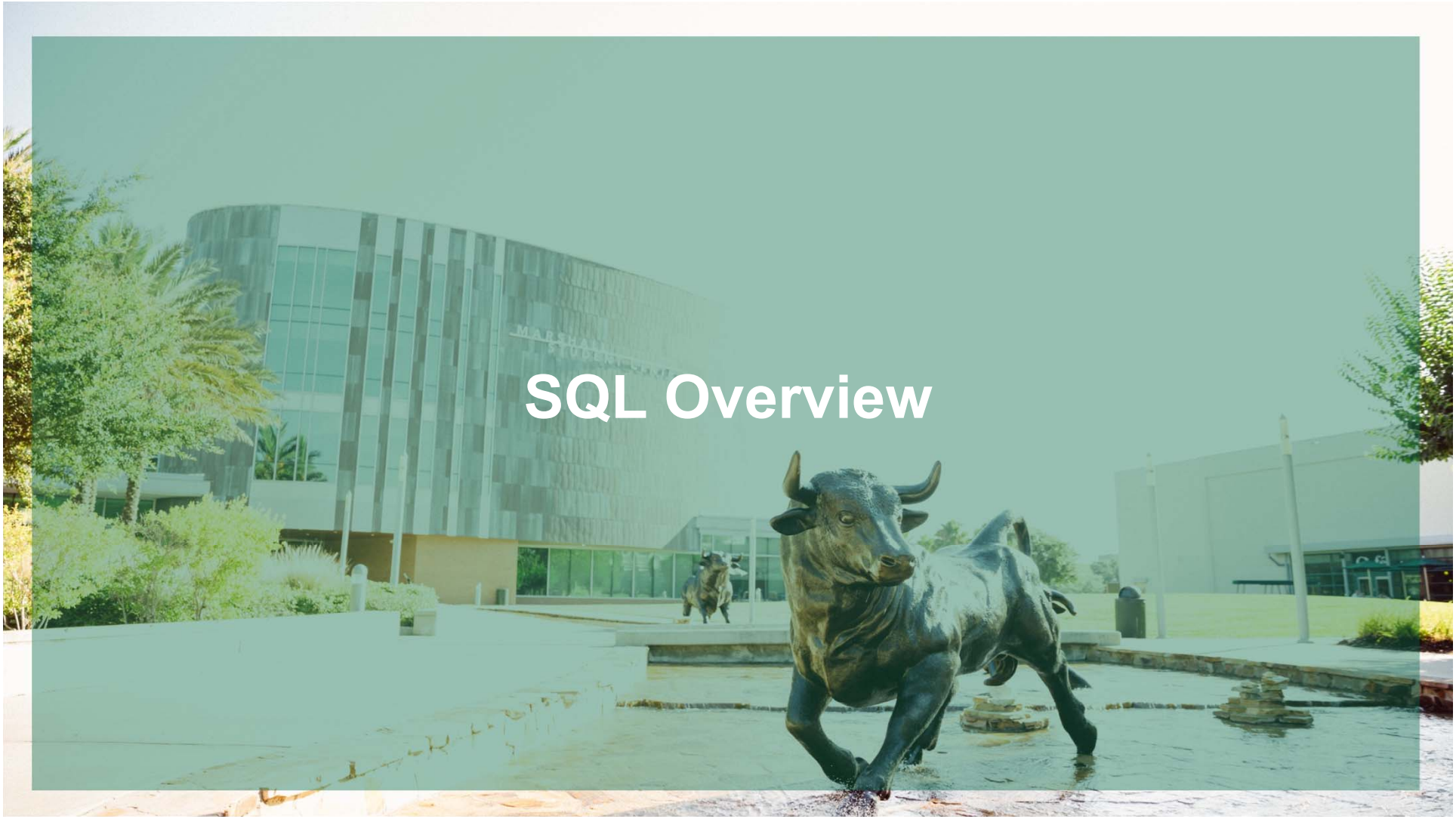
Database Schema (Continued)



- **Relationship Types (also known as cardinality):**
  - **One-to-many relationship** – a student can take one or many courses.
  - **Many-to-many relationship** – Many students can take many courses.
- **Database normalization** will eliminate the many-to-many relationships as it is inefficient to store this data due to data duplication.

Database Key Terms

- **Schema** – Represents a catalog of tables. A single database can have one or more schemas.
- **Primary Key** – The unique identifier within a given table. Can be one or more fields (known as a composite key).
- **Foreign Key** – A key used outside of its origin. You will see this as part of the normalization process.
- **Index** – Used to optimize how the table will be queried.
- **Constraints** – Rules established within the database to ensure data integrity (e.g., column cannot be NULL.
- **Synonyms** – Alternative name assigned to a database object.
- **Views** – Predefined code to create a virtual table (examples are most of the objects that we retrieve from the Data Warehouse).
- **Stored Procedures** – Predefined code that will perform database activity. An example of this could be a monthly purge of old documents or an overnight batch process.
- **Transaction Log** – All database activity is initially stored in this log to ensure that the activity is properly performed. In the event of a failed transaction, the transaction can be rolled back through this log.


SQL Overview

SQL Overview

- SQL is used to insert, retrieve, modify, and delete data within a database.

- SQL is divided into the following categories:
  - **Data Definition Language (DDL)** – Defines the data objects (e.g., tables, columns, keys, indexes, etc.) within the database based on the defined data model.
  - **Data Manipulation Language (DML)** – Used to insert, update, or delete data within a database.
  - **Data Query Language (DQL)** – Used to query or select data from a database based on the defined data model. **This training will only focus on this category.**

SQL Overview (Continued)

- SQL follows a standard created by the American National Standards Institute (ANSI) / International Organization for Standardization (ISO).
- Interactions are periodically made to reflect the needs of database users. Additionally, each database vendor adds proprietary language to enhance the ANSI/ISO standards to improve the customer experience (although this makes it difficult to change database vendors if proprietary language is used).
- First iteration from ANSI/ISO was known as SQL-89 (SQL1). The latest released version is ISO/IEC 9075:2011 (the seventh revision of the language). USF tends to follow SQL-89.

SQL Syntax – Single Table

- All SQL queries begin with the word **SELECT**. This tells the database that a read-only statement will be executed.
- Once SELECT is specified, the specific **COLUMNS** are specified as to which data points are being pulled.
- Once all the COLUMNS are specified, the **FROM** clause is specified. This tells the database what table to pull the data from. The schema and table name is specified after the FROM is specified.
- Once the FROM table has been specified, the **WHERE** clause is specified which describes the specific filters being applied to the data.
- If data aggregation is being performed (similar to the summarization option within ACL), a **GROUP BY** clause is used to tell the database how to group the objects.
- If data aggregation is being used and filters need to be used for aggregated data, the **HAVING** clause is specified.
- If you would like the result set to be organized in a certain way, the **ORDER BY** clause is used. Either the column name is used or the column order in which it is represented.

SQL Syntax – Single Table (Continued)

- Within the **SELECT** statement, columns can be aliased within the query. For example, if the technical object is Student_ID, you can use an alias to say Student ID instead.
  - SELECT Student_ID AS 'Student ID'
- The **WHERE** clauses commonly use the following operators:
  - = - Exact match based on what is listed after (WHERE student_id = 1)
  - IN - Used for a series (WHERE student_id IN (1,2))
  - LIKE – Used to do a keyword match (WHERE first_name LIKE 'B%')
  - BETWEEN – Used to return a range of values (WHERE date BETWEEN '01/01/2020 AND '01/31/2020').
  - AND – Used to specify more than a single statement that must be met.
  - OR – Used to specify an alternate condition that must be met.
  - Parenthesis are used to group AND/OR statements to only focus on specific combinations.

SQL Syntax – Single Table (Example 1)

- SELECT first_name, last_name, student_id, address
- FROM dbo.student_tbl
- WHERE first_name LIKE ‘a%’
- ORDER BY 1;

SQL Syntax – Single Table (Example 2)

- SELECT student_id, count(student_id) AS ‘Count’
- FROM dbo.student_tbl
- WHERE first_name LIKE ‘a%’
- GROUP BY student_id
- HAVING count(student_id) > 1
- ORDER BY student_id;

SQL Syntax – Single Table (Example 3)

- SELECT first_name, last_name, student_id, address
- FROM dbo.student_tbl
- WHERE (first_name LIKE ‘A%’ AND last_name LIKE ‘A%’)
- OR student_id = 1

SQL Syntax – Order of Operations

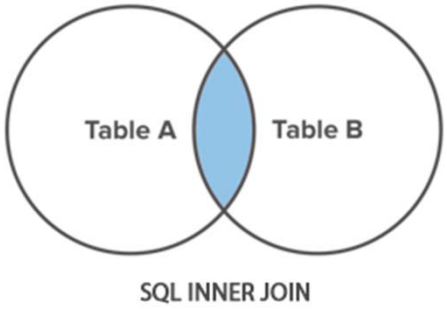
- The database will read the query in the following order:
  - FROM
  - JOIN(s)
  - WHERE
  - GROUP BY
  - HAVING
  - SELECT
  - DISTINCT
  - ORDER BY
  - LIMIT

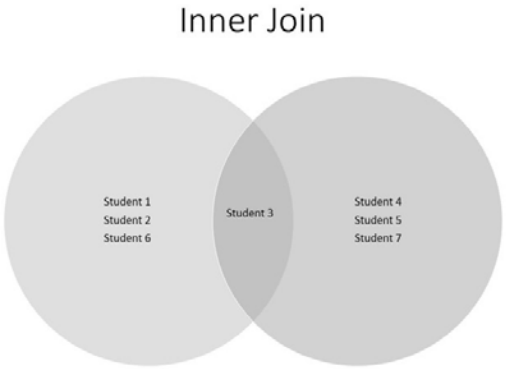
SQL Syntax – Joins

- Since tables are normalized, it is common for **database joins** to be required. These are used to combine two or more tables based on the keys within each table (e.g., linking a primary key to it's foreign key within another table).

- There are three types of joins:
  - **INNER JOIN** – Will only return objects that are matched.
  - **OUTER JOIN** – Depending on the primary table selected, will return matched objects from secondary table and all objects in the primary table.
    - **LEFT OUTER JOIN** – Uses the first table (one specified in the FROM) as the primary table.
    - **RIGHT OUTER JOIN** – Uses the second table specified in the JOIN statement.
    - **FULL OUTER JOIN** – Returns all results from both tables regardless if there are matched on either side. Where there are matches, the linked values will be included.
  - **CROSS JOIN** (also known as a cartesian join) – Will match all values in table A to table B together. This is very uncommon in practice.

SQL Syntax – Joins – Inner Join Illustrated





SQL Syntax – Inner Join SQL

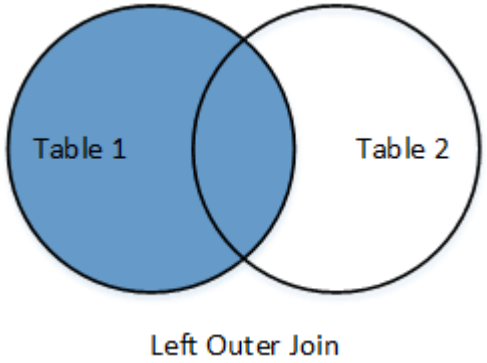
Under SQL-89

```
SELECT product_name,category_name,
list_price
FROM production.products p
,production.categories c
WHERE c.category_id =  p.category_id
ORDER BY product_name DESC;
```

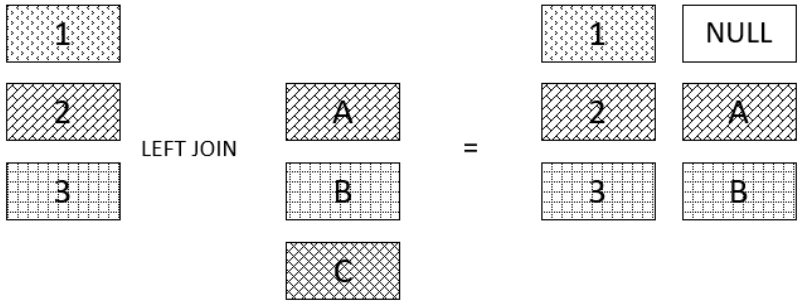
Under SQL-92

```
SELECT product_name,category_name,
list_price
FROM production.products p
INNER JOIN production.categories c
    ON c.category_id =  p.category_id
ORDER BY product_name DESC;
```

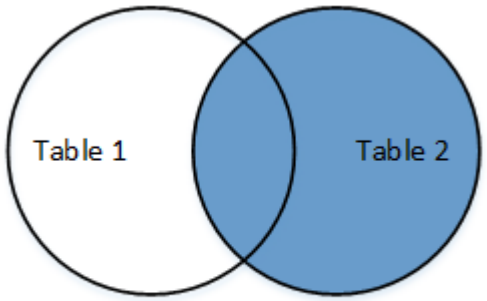
SQL Syntax – Joins – Left Outer Join Illustrated



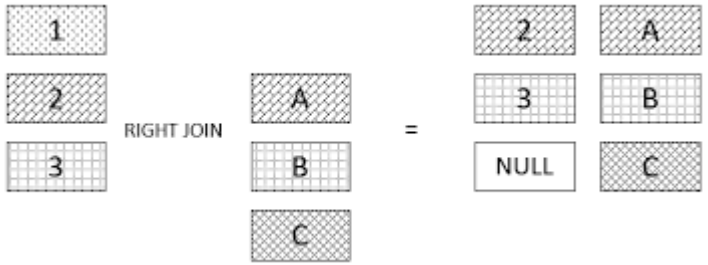
Left Outer Join



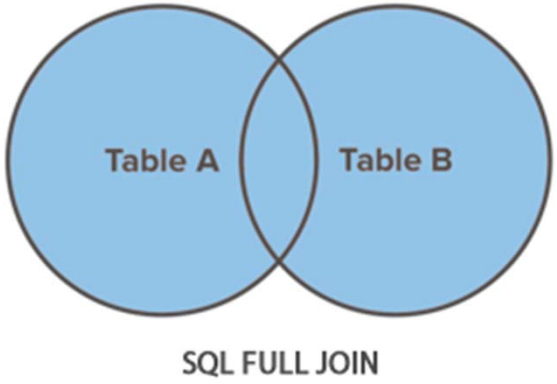
SQL Syntax – Joins – Right Outer Join Illustrated

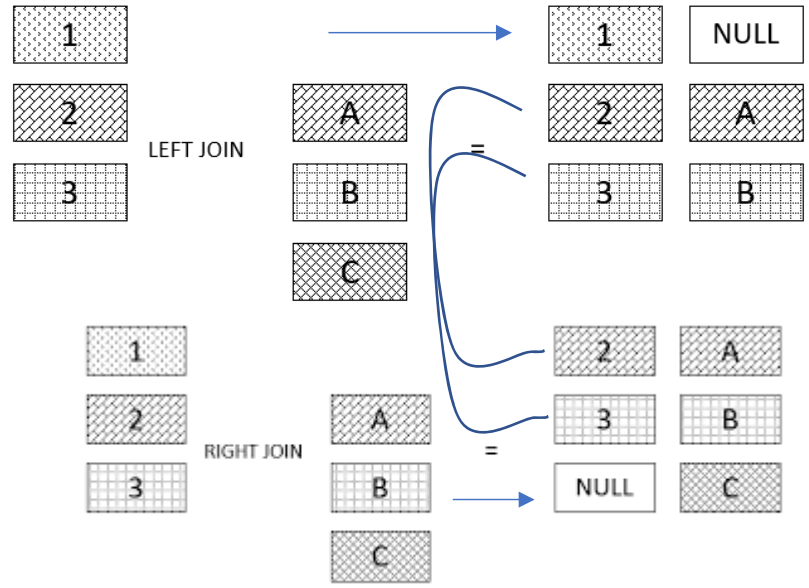


Right Outer Join



SQL Syntax – Joins – Full Outer Join Illustrated





SQL Syntax – Left Outer Join SQL

Under SQL-89

```
SELECT product_name,category_name,
list_price
FROM production.products p
,production.categories c
WHERE c.category_id =
p.category_id(+)
ORDER BY product_name DESC;
```

Under SQL-92

```
SELECT product_name,category_name,
list_price
FROM production.products p
LEFT OUTER JOIN
production.categories c
    ON c.category_id =  p.category_id
ORDER BY product_name DESC;
```

SQL Syntax – Right Outer Join SQL

Under SQL-89

```
SELECT product_name,category_name,
list_price
FROM production.products p
,production.categories c
WHERE c.category_id(+) =
p.category_id
ORDER BY product_name DESC;
```

Under SQL-92

```
SELECT product_name,category_name,
list_price
FROM production.products p
RIGHT OUTER JOIN
production.categories c
    ON c.category_id =  p.category_id
ORDER BY product_name DESC;
```

SQL Syntax – Full Outer Join SQL

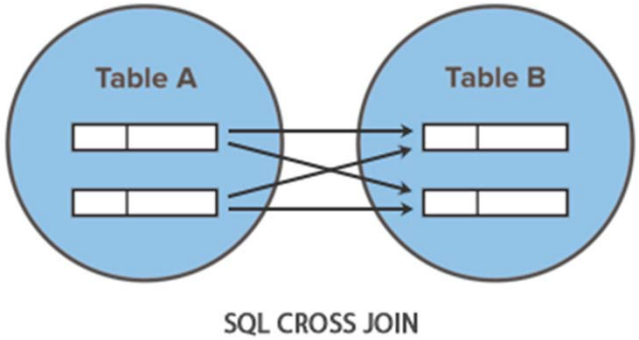
Under SQL-89

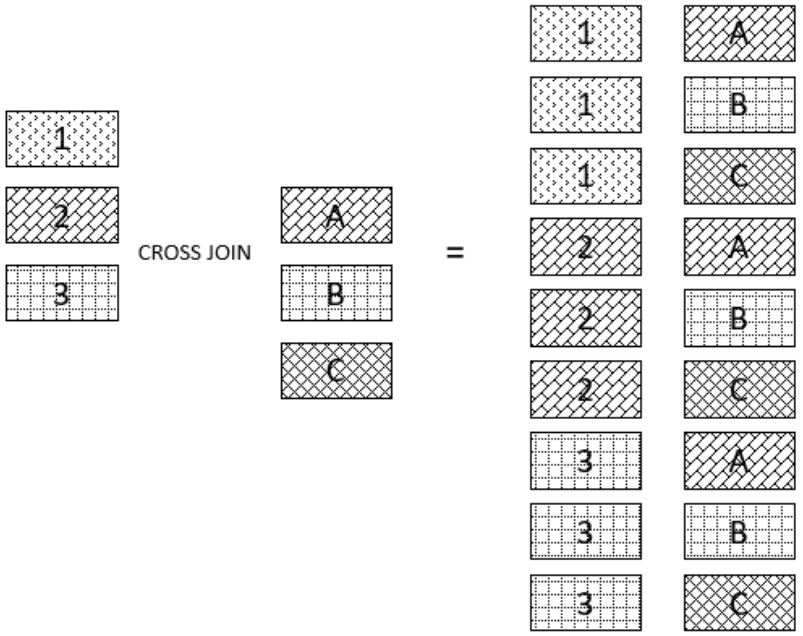
Not applicable

Under SQL-92

```
SELECT product_name,category_name,
list_price
FROM production.products p
FULL JOIN production.categories c
    ON c.category_id =  p.category_id
ORDER BY product_name DESC;
```

SQL Syntax – Joins – Cross Join Illustrated





SQL Syntax – Cross Join SQL

Under SQL-89

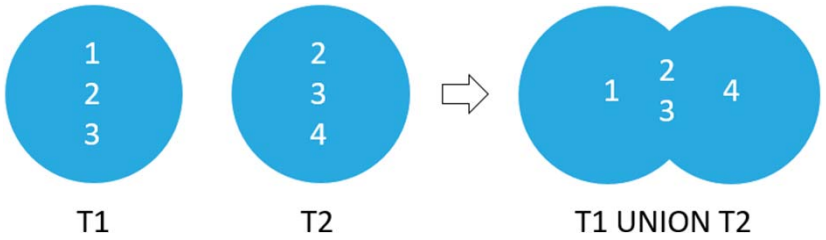
```
SELECT product_name,category_name,
list_price
FROM production.products,
production.categories;
```

Under SQL-92

```
SELECT product_name,category_name,
list_price
FROM production.products
CROSS JOIN production.categories;
```

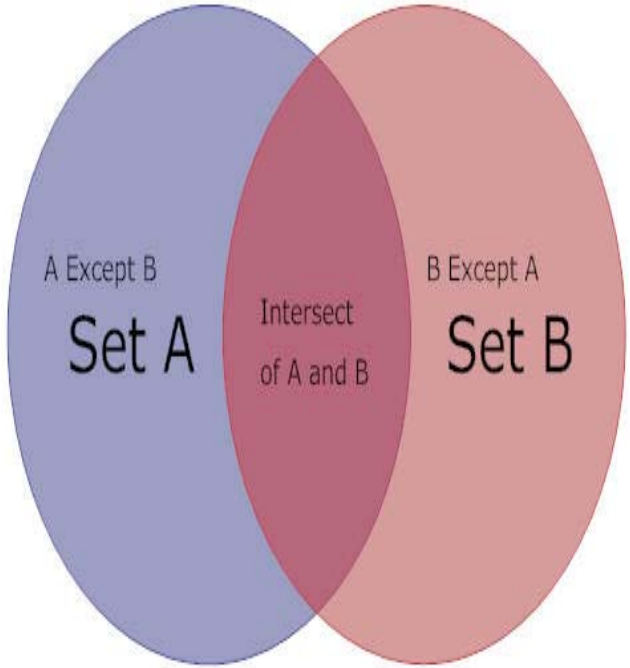
SQL Syntax – Unions

- **Union** – Combines a two or more tables into a single result. The columns specified must have the same data types and number of columns. Within SQL, the commands are as follows:
  - **UNION ALL** – Combined all values among two tables or more.
  - **UNION** – Combines only distinct values within the tables.



SQL Syntax – Intersect and Except

- **Intersect** – Returns results where values are matched as a single result set.
- **Except** – Returns the difference between two result sets.



SQL Syntax – Union, Intersect, and Except

```
SELECT [Student_ID]
FROM [dbo].[Students]

UNION (other options here are UNION ALL, INTERSECT, and EXCEPT)

SELECT [Student_ID]
FROM [dbo].[Students2]
```

SQL Syntax – Subquery

- **Subquery** – a nested query within a SQL statement.

```
SELECT order_id, order_date, customer_id
FROM sales.orders
WHERE customer_id IN (
        SELECT customer_id
        FROM sales.customers
        WHERE city = 'New York'
    )
```

SQL Syntax – Correlated Subquery

- **Correlated Subquery** – a type of subquery that performs a join operation to return a single result set.

```
SELECT product_name, list_price, category_id
FROM production.products p1
WHERE list_price IN (
        SELECT MAX (p2.list_price)
        FROM production.products p2
        WHERE p2.category_id = p1.category_id
        GROUP BY
            p2.category_id
    )
```

SQL Syntax – Exists

- **Exists –** Used to return a TRUE or FALSE statement (also known as a binary result). In practice, functions very similar to the IN condition of a WHERE statement.

```
SELECT *
FROM sales.orders o
WHERE
    EXISTS (SELECT customer_id
        FROM sales.customers c
        WHERE o.customer_id = c.customer_id
        AND city = 'San Jose'
    )
```

SQL Syntax – Common Table Expressions

- **Common Table Expressions (CTEs)** are an advanced form of SQL used to perform recursive functionality. Recursive functions typically increment a value until a certain goal is obtained. An example would be creating a organizational hierarchy that shows all the reporting relationships.

- These types of statements begin using the **WITH** operator.

- This topic is complicated and will not be included for this discussion.

UNIVERSITY of
SOUTH FLORIDA