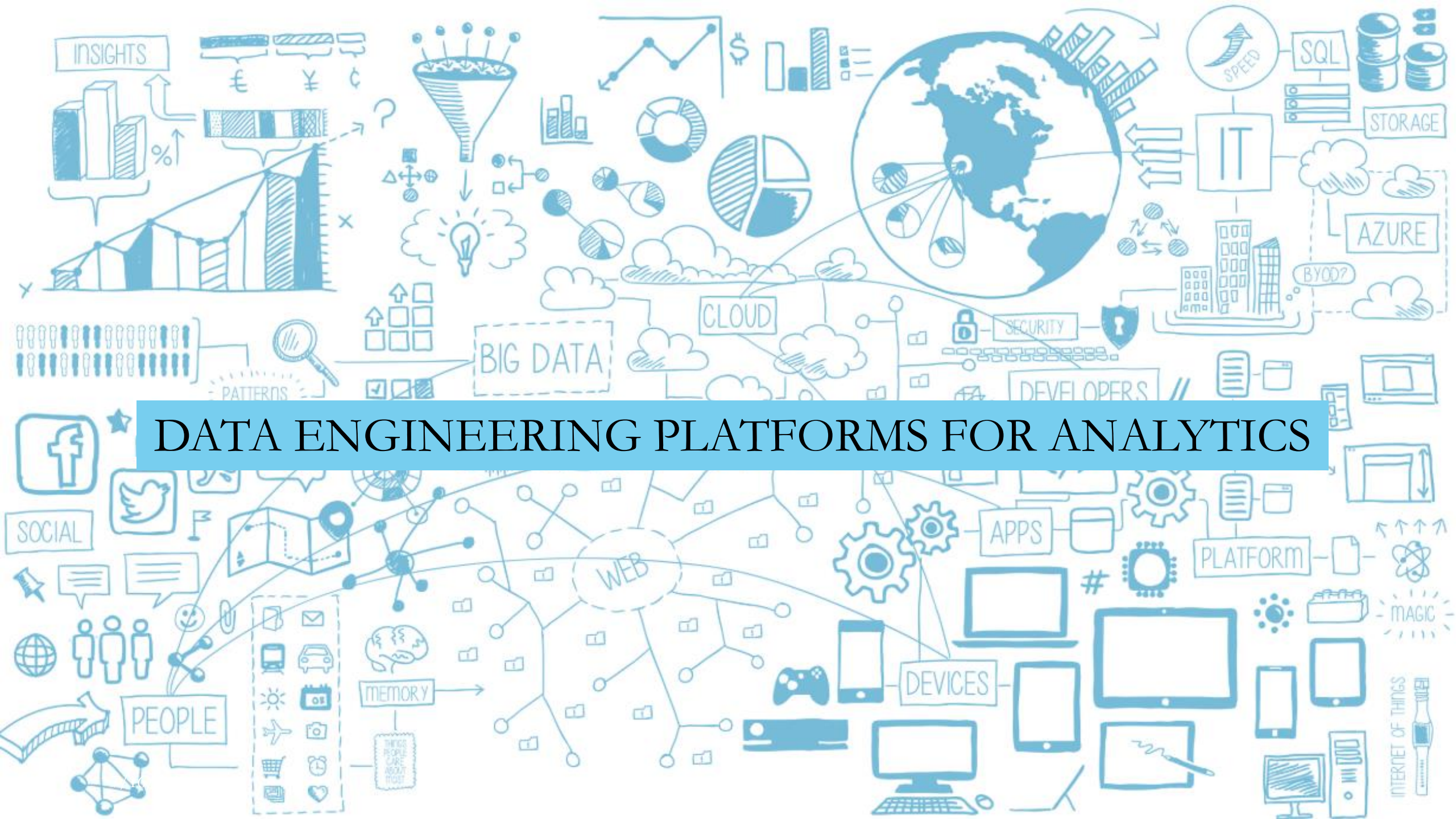


DATA ENGINEERING PLATFORMS FOR ANALYTICS

Syllabus

- Session 1 – Foundations of data systems
- Session 2 – Relational Databases
- Session 3 – Structured Query Language(SQL)
- Session 4 – Advanced SQL
- Session 5 – Analytical data platforms

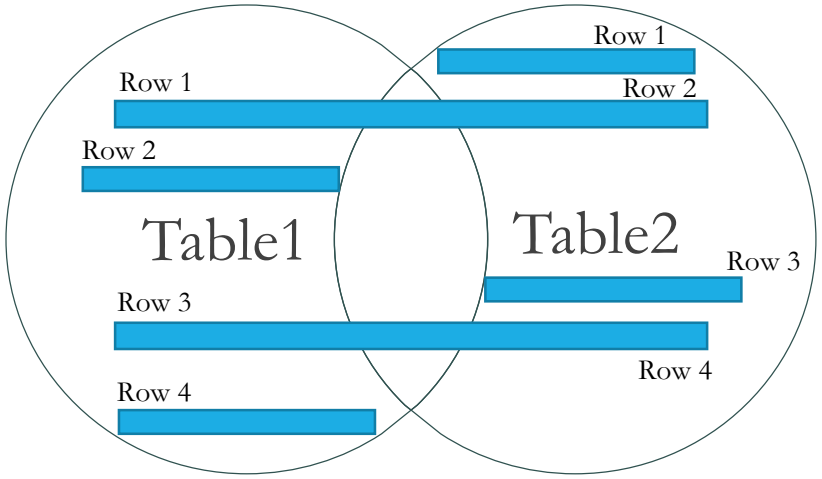
Syllabus

- Session 6 – Business Intelligence
- Session 7 – Data Pipelines (GCP)
- Session 8 – Document & Graph databases
- Session 9 – Columnar, Key Value DB & Blockchain
- Session 10 – Final Exam & Project Presentations

Combining Data

SQL Joins

- SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them
- E.g. Relational tables Table 1 and Table 2 related by PK, FK



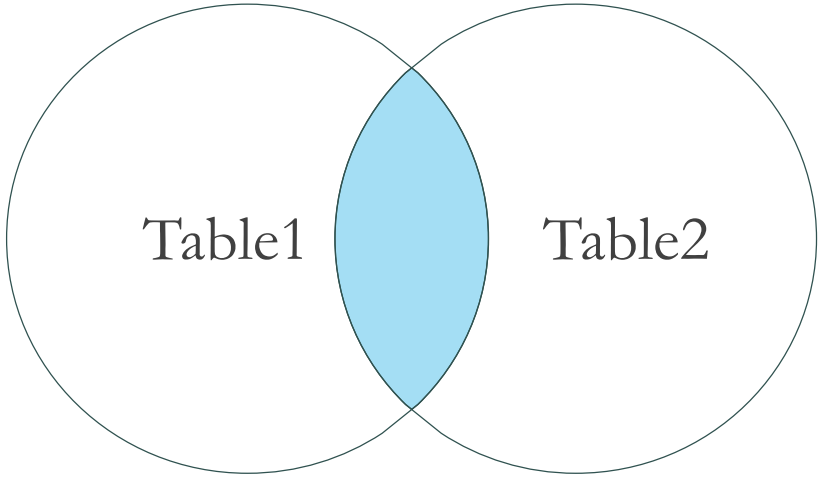
https://codebabes.com/blog/getting-freaky-mysql-joins/

Types of Joins

- Inner Join
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
- Cross Join

Inner (Simple) Join

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.
- Most common type of join




```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

Left Outer Join

- Returns all rows from the left table (Table1), with the matching rows in the right table (Table2).
- The result is NULL in the right side when there is no match.

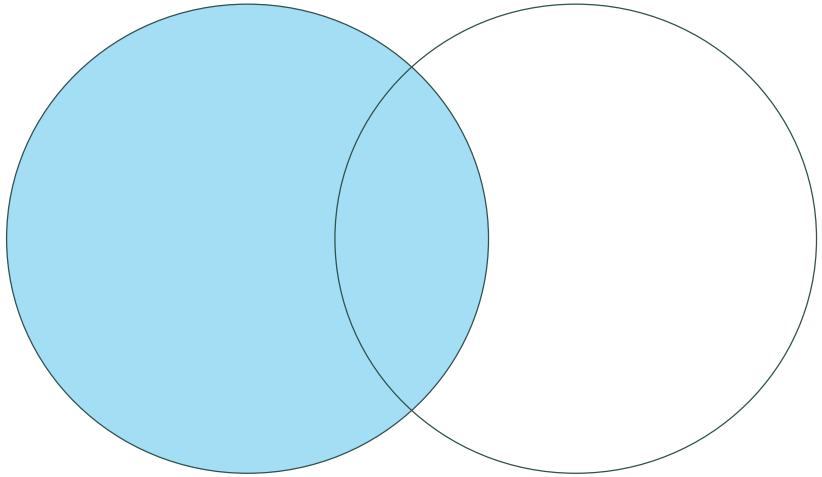


Table1 Table2

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

Right Outer Join

- Returns all rows from the right table (Table2), with the matching rows in the left table (Table1).
- The result is NULL in the left side when there is no match.

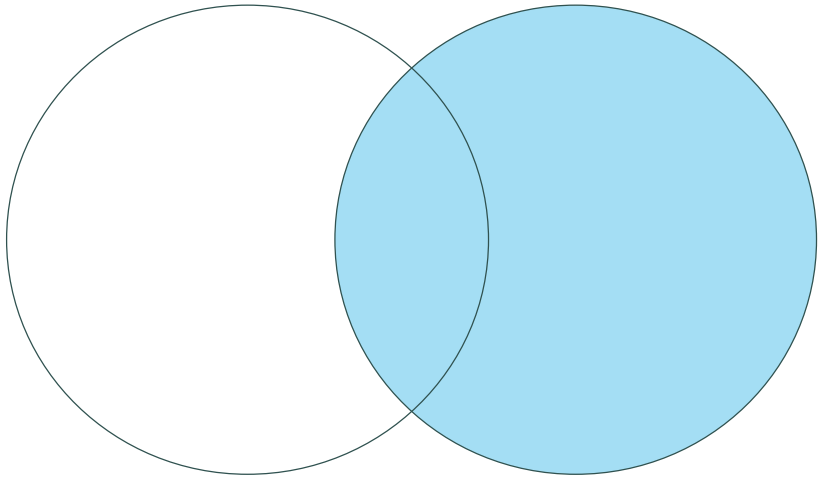


Table1 Table2

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name;
```

Full Outer Join

- Returns all rows from the left table (Table1) and from the right table (Table2).
- Combines the result of both LEFT and RIGHT joins.

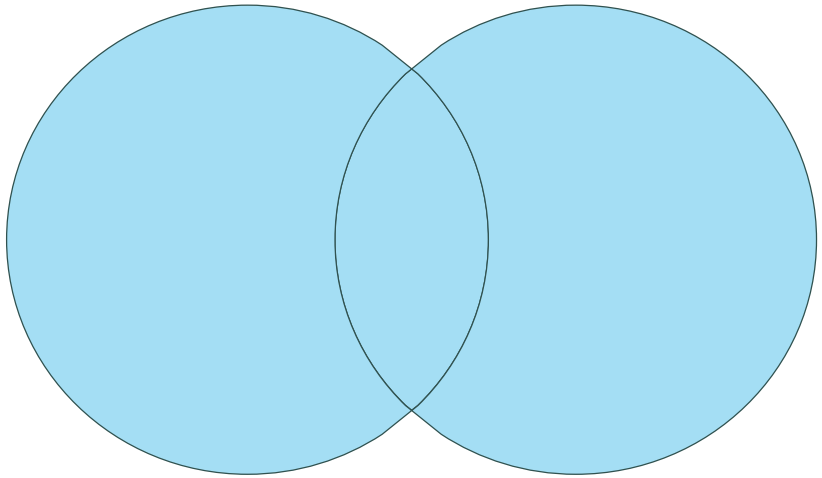


Table1 Table2

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

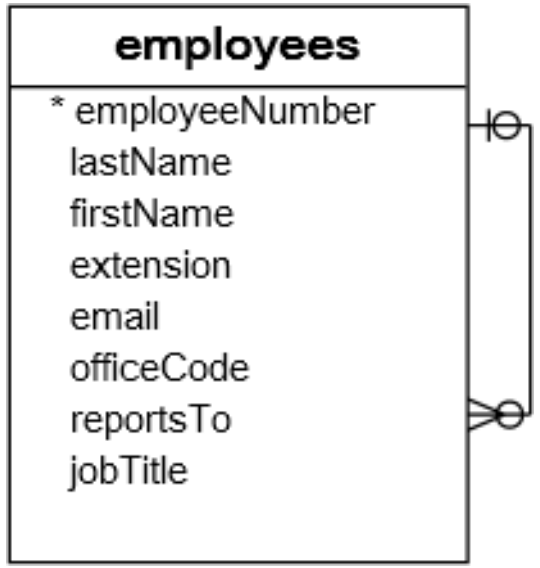
Cross (Cartesian) Join

- Cartesian product of the tables that involved in the join
- Returns # rows Table1 x # rows Table 2
- Could lead to performance issues due to large result set

```
SELECT column_name(s)
FROM table1 t1,
CROSS JOIN table2 t2
```

Self Join

- Used to join a table to itself, as if the table were two tables
- E.g. Employees table org structure report




```
SELECT column_name(s)
FROM table1 t1,
INNER JOIN table1 t2
ON t1.column1 = t2.column2
```

Union

- Used to combine two or more result sets from multiple tables into a single result set
- The number of columns in the corresponding select statements must be equal and of the same/convertible data type.
- Eliminates duplicate rows from by default

```
SELECT column_name(s)
UNION [DISTINCT | ALL]
SELECT column_name(s)
UNION [DISTINCT | ALL]
```

Combining Data - SQL

1. Inner Join
2. Left Join
3. Right Join
4. Outer Join
5. Natural Join
6. Self Join
7. Union

Exercise 1

Combining Data

Nested Queries

Nested Query

- Query that is nested inside another query
- Also called an Inner query or Subquery while the query that contains it is called an Outer Query

SELECT column_name(s) FROM table_name WHERE column_name operator value	OPERATOR **Inner Query**
Outer Query	(SELECT column_name FROM table_name WHERE column_name operator value)

Nested Query Rules

- Nested query must always appear within parentheses
- Possible to have as many level of nesting as needed
- Can be nested inside SELECT, INSERT, DELETE, or UPDATE
- May occur in SELECT, FROM, WHERE and HAVING

Advantages of Nested Queries

- Allow the use of results of another query in the outer query
- Complex queries can be broken down into a series of logical steps for easy understanding and code maintenance
- Can replace complex joins and unions in some cases

Nested Query - SQL

1. Nested queries with IN and NOT IN operators
2. Nested queries with EXISTS and NOT EXISTS
3. Nested query in FROM clause
4. Nested query in SELECT statement

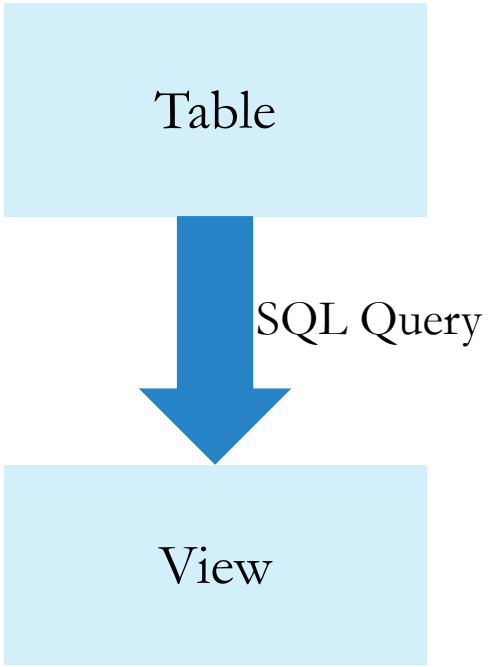
Exercise 1

Nested Queries

Views and Indexes

View

- A virtual table that doesn't require physical storage
- Also called a named query or a stored query
- Data can be manipulated in a view, with some restrictions



Advantages of Views

- Used to simplify data access
  - In large multi-user systems, views make it easy to control access to data for different types of users
- Used to maintain data security
  - Used to restrict user access to particular columns or rows in a table
- Used to maintain summarized data
  - Simplifies select statements for users and applications

Restrictions on Views

- Creation of views:
  - The SELECT statement cannot contain a subquery in the from clause
  - The SELECT statement cannot refer to system or user variables
  - A temporary view cannot be created or
  - A view cannot be associated with a trigger
  - Rows in the view and the rows in the underlying table must have 1:1 relationship
- Update of views:
  - Aggregate Functions, DISTINCT, GROUP BY, HAVING, UNION, Subquery in SELECT, certain joins etc.
  - Refer to MySQL documentation

Create and Drop View

- View can be created from one or more tables
- View be deleted like other database objects
- View may also be created from another view

```
CREATE OR REPLACE VIEW view_name AS
    SELECT <statement>;
```

```
DROP VIEW view_name;
```

```
CREATE VIEW view1
AS SELECT * FROM  view2
```

Update View

- Rows in the view and the rows in the underlying table must have 1:1 relationship
- Views containing the following are not updatable:
  - Aggregate Functions, DISTINCT, GROUP BY, HAVING, UNION, Subquery in SELECT, certain joins etc.
  - Refer to MySQL documentation:

    http://dev.mysql.com/doc/refman/5.7/en/view-updatability.html

Materialized Views

- Materialized view creates and stores the result table in advance, filled with data
- The scheme of this table is given by the SELECT clause of the view definition
- Useful when the query involves many joins of large tables, or any query that has long execution times
- Not supported by MySQL.

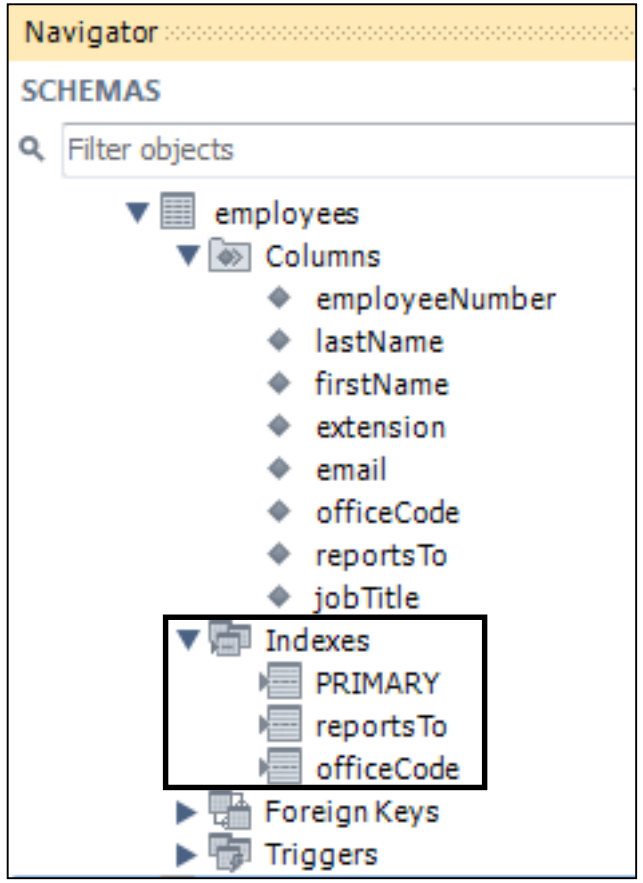
Indexes

- Data structures used to find records within a table more quickly
- Indexes consist of rows and columns
- Built on one or more columns of a table
- Indexes are not supported for Views

```
CREATE INDEX index_name ON table_name (column_name(s));
CREATE UNIQUE INDEX index_name ON table_name (column_name(s));
DROP INDEX index_name;
```

Indexing

- Data structure technique to efficiently retrieve records from the database
- Based on attributes on which the indexing has been done
- Each index is associated with only one table



Types of Indexing

- Clustered Index
  - stores row data in order
  - efficient when data is sorted
- Non-Clustered Index
  - data is arranged in a random way
  - index order not the same as physical ordering of data
  - used for ORDER BY, WHERE and JOIN statements

Advantages & Limitations

- Indexes are best used on columns that are frequently used in where clauses, and in any kind of sorting, such as "order by".
  - Indexes slow down inserts/updates, use them carefully on frequently updated columns
  - Indexes speed up where clauses and order by. Remember to think about HOW your data is going to be used when building your tables. There are a few other things to remember. If your table is very small, i.e., only a few employees, it's worse to use an index than to leave it out and just let it do a table scan.
  - Indexes really only come in handy with tables that have a lot of rows.
  - Don't forget joins too! Indexed join fields speed things up.

Views SQL

1. Create views with SELECT statements
2. Update data in underlying tables by updating views
3. Enforce data integrity using CHECK OPTION
4. Delete views
5. Create and delete indexes

Exercise 1

Views & Indexes

Transforming Data

String Functions

String Functions	Description
LOWER() /UPPER()	Returns the argument in lowercase/uppercase
TRIM()	Removes leading and trailing spaces
LTRIM() /RTRIM()	Removes leading/trailing spaces
LPAD()/ RPAD()	Returns the string argument, left-padded/right-padded with the specified string
CONCAT()	Returns concatenated string
SUBSTRING(), SUBSTR()	Returns the substring
LOCATE(),POSITION()	Returns the position of the first occurrence of substring

Numeric Functions

Numeric Functions	Description
ABS()	Returns the absolute value of numeric expression
CEIL()	Returns the smallest integer value that is not less than passed numeric expression
FLOOR()	Returns the largest integer value that is not greater than passed numeric expression
LOG()	Returns the natural logarithm of the passed numeric expression
EXP()	Returns e (based of natural log) raised to the power of passed numeric expression
POW()	Returns the value of one expression raised to the power of another expression
ROUND()	Used to round an expression to a number of decimal points
SQRT()	Returns the non-negative square root of numeric expression

Date Functions

Date Functions	Description
NOW()	Returns the current date and time
CURDATE()	Returns the current date
DATE()	Extracts the date part of a date or datetime expression
DATEDIFF()	Subtracts two dates
DATE_ADD()	Adds two dates
SYSDATE()	Returns the time at which the function executes
DAYOFMONTH()	Returns the day of the month (1-31)
DAYOFWEEK()	Returns the weekday index of the argument
DAYOFYEAR()	Returns the day of the year (1-366)

Data Conversion - Implicit

- Converts a value of any type into a value that has a specified type
- Target type can be any one of the following types: BINARY, CHAR, DATE, DATETIME, TIME,DECIMAL, SIGNED, UNSIGNED.
- MySQL automatically converts some data types for comparison

```
mysql> SELECT 1+'1';
-> 2
```

Data Conversion – Explicit (CAST, CONVERT)

- Converts a value of any type into a value that has a specified type
- CONVERT() with USING converts data between different character sets

```
CAST(expression AS TYPE);
CONVERT(expression, type);
CONVERT(expr USING transcoding_name);
```

E.g - CAST ($210.27 AS VARCHAR(10))

Transforming Data - SQL

1. String Functions
2. Numeric Functions
3. Date Functions
4. Convert data types using CAST

Exercise 1

Transforming Data

Advanced Topics

User-defined variables

- Enables passing a value from an SQL statement to another SQL statement
- The maximum length of the user-defined variable is 64 characters and they are not case sensitive
- An user-defined variable is session-specific.

Transactions (TCL)

- A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit.
- Most of the data definition statements (DDL), such as CREATE, ALTER, DROP database/tables cannot be used within a transaction.

Transactions (Continued)

- Transactions have ACID properties and below commands are used to execute a transaction
  - START TRANSACTION – Start a transaction
  - COMMIT – persist the work done
  - ROLLBACK - restore to original state since the last commit
  - END TRANSACTION – End a transaction

Temporary Tables

- A temporary table is very handy when it is impossible or expensive to query data that requires a single SELECT statement with JOIN clauses. In this case, you can use a temporary table to store the immediate result and use another query to process it.
    - Use the DROP TABLE statement to remove a temporary table explicitly.
    - A temporary table is only available and accessible to the client that creates it. In the same session, two temporary tables cannot share the same name.
    - A temporary table can have the same name as a normal table in a database

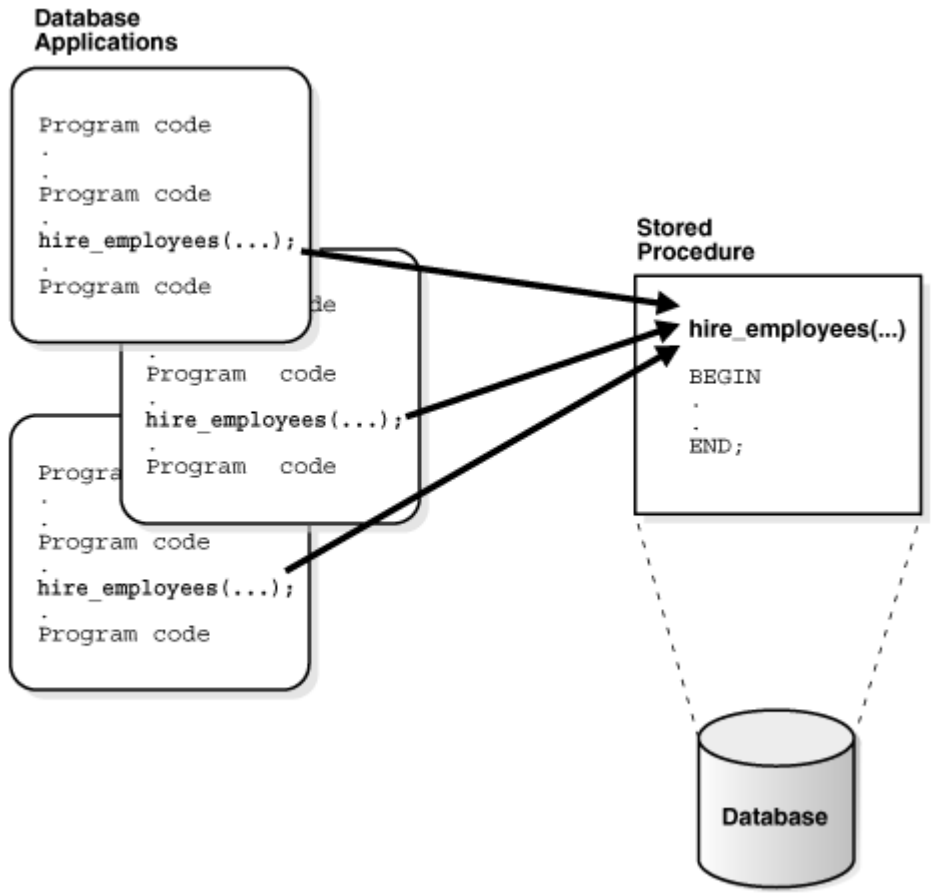
DCL – Data Control Statements

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with GRANT

GRANT SELECT, INSERT, UPDATE, DELETE ON *table_name TO user_name*;
REVOKE INSERT ON *table_name TO user_name*;

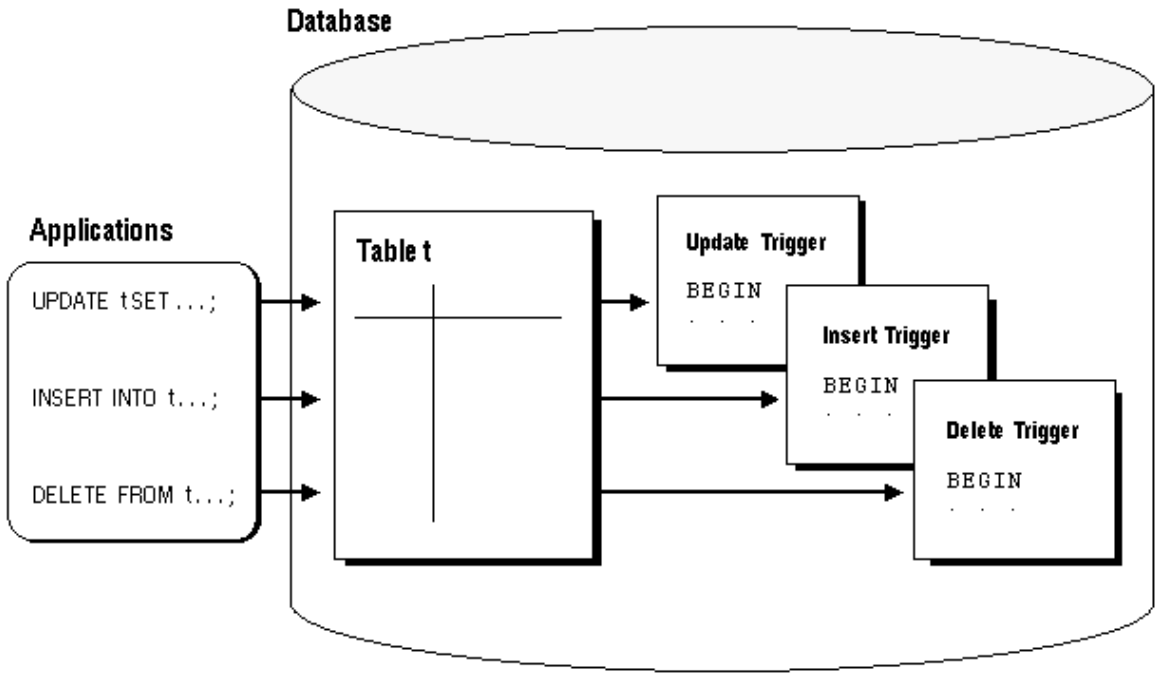
Stored Procedures

- In a database management system (DBMS), a Stored Procedure is a set of SQL statements with an assigned name that's stored in the database in compiled form so that it can be shared by a number of programs.



Triggers

- A trigger is a special type of stored procedure
- A trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly



Partitioning

- Partitioning enables tables and indexes or index-organized tables to be subdivided into smaller manageable pieces and these each small piece is called a "partition".

oracle-base.com

Advantages of partitioning

- Smaller and manageable pieces of data(Partitions)
- Reduced recovery time
- Failure impact is less
- import / export can be done at the " Partition Level".
- Faster access of data
- Partitions work independent of the other partitions.
- Very easy to use

Sequence

- A list of integers generated in the ascending order.
- Many applications need sequences to generate unique numbers mainly for identification e.g., customer ID in CRM, employee numbers in HR, and equipment numbers in the services management system.
- The following rules are applied when you use the AUTO_INCREMENT attribute:
  - Each table has only one AUTO_INCREMENT column whose data type is typically the integer.
  - The AUTO_INCREMENT column must be indexed, which means it can be either PRIMARY KEY or UNIQUE index.
  - The AUTO_INCREMENT column must have a NOT NULL constraint. When you set the AUTO_INCREMENT attribute to a column, MySQL automatically adds the NOT NULL constraint to the column implicitly.

Full-Text Search

- MySQL supports text searching by using the LIKE operator and Regular Expressions. However, when the text column is large and the number of rows in a table is increased, using these methods has some limitations:
  - Performance: MySQL has to scan the whole table to find the exact text based on a pattern in the LIKE statement or pattern in the regular expressions.
  - Flexible search: with the LIKE operator and regular expression searches, it is difficult to have a flexible search query e.g., to find product whose description contains car but not classic.
  - Relevance ranking: there is no way to specify which row in the result set is more relevant to the search terms

Natural Language full-text searches

- Rows or documents that are relevant to the free-text natural human language query.
  - Relevance is a positive floating-point number.
    - Relevance = 0, No similarity.
    - Relevance based on various factors including the number of words in the document, the number of unique words in the document, the total number of words in the collection, and the number of documents (rows) that contain a particular word.
  - To perform natural language full-text searches, use
    - MATCH() function - specifies the column where you want to search
    - AGAINST() functions - determines the search expression to be used.

Defining Full Text Indexes

- Before performing a full-text search in a column of a table, you must index its data. MySQL will recreate the full-text index whenever the data of the column changes. In MySQL, the full-text index is a kind of index that has a name FULLTEXT.

```
CREATE TABLE table_name(
        column1 data_type,
        column2 data_type,
        column3 data_type,
…
PRIMARY_KEY(key_column),
FULLTEXT (column1,column2,..));
```

http://www.mysqltutorial.org/

Appendix

References

- http://www.mysqltutorial.org/
- http://www.techonthenet.com/mysql/
- http://stackoverflow.com/
- http://www.tutorialspoint.com/