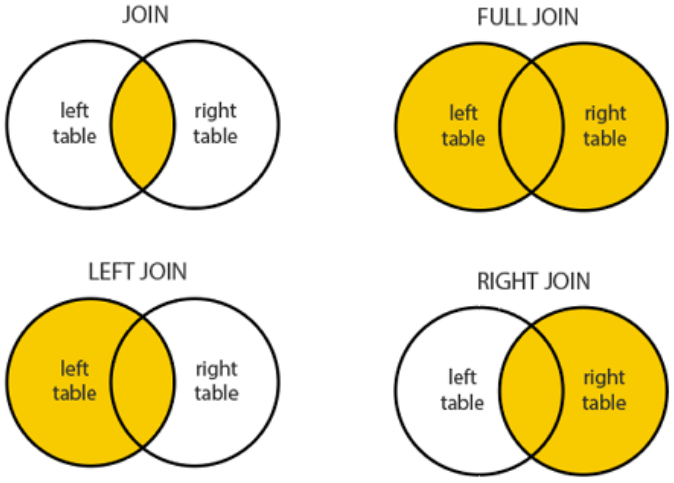
1 Basic Commands

- CREATE TABLE: Creates a table
- INSERT: Inserts a row (or set of rows) into a given table
- UPDATE: Modifies already-existing data
- DELETE: Removes a row (or groupe of rows)
- SELECT: Selects certain columns from a table
- GROUP BY: Groups rows having contents of a column
- WHERE: Filter **before** any grouping is applied
- HAVING: Filter **after** any grouping is applied
- ORDER BY: Sorts results by column(s)
- DISTINCT: Returns only distinct values
- UNION: Combines results from multiple tables
- DATE_TRUNC(*month*, *timestamp*): extract month from timestamp

2 Joins

- OUTER JOIN: Combine and preserve all rows
- LEFT JOIN: Combine and preserve rows from left table
- RIGHT JOIN: Combine and preserve rows from right table
- JOIN: Combine and preserve rows from both tables




```sql
SELECT
    COUNT(DISTINCT u.user_id)
FROM
    users u
    JOIN posts p ON u.user_id = p.user_id
```

3 Common Table Expressions (CTEs) and Subqueries

- CTEs: Define a table and allow it to be referenced later using an alias (i.e distribution of posts by users)

```sql
WITH user_post_count AS (
    SELECT
        users.user_id,
        COUNT(post_id) AS num_posts
    FROM users
    LEFT JOIN posts on users.user_id = posts.user_id
    GROUP BY 1
)

SELECT num_posts, COUNT(*) as num_users
FROM user_post_count
GROUP BY 1
```

- Subqueries: are inline in the query itself and must have a unique alias

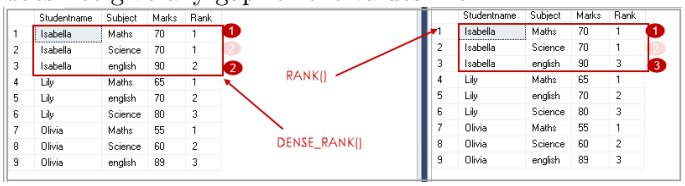
```sql
SELECT num_posts, COUNT(*) as num_users
FROM
    (
    SELECT users.user_id,
        COUNT(post_id) as num_posts
    FROM users
        LEFT JOIN posts on users.user_id = posts.user_id
    GROUP BY 1
    ) u
```

4 Window Functions

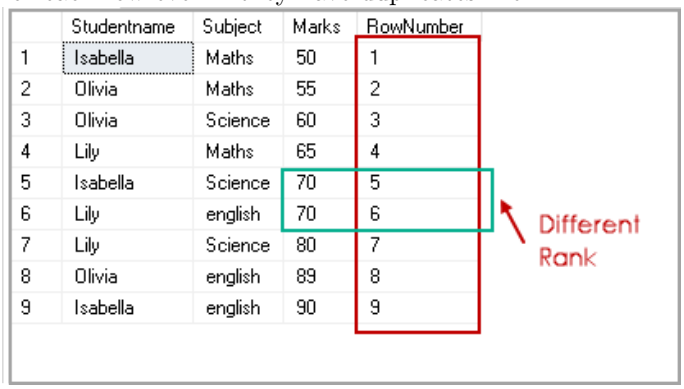
- Window Functions: Performs a calculation across a set of rows. This can be done by an aggregation function. But unlike regular aggregate functions, a window function **does not** cause rows to become grouped into a single output row - the rows retain their separate identities. i.e..

```sql
SELECT
    first_name, last_name, department_id,
    ROUND(AVG(salary) OVER (PARTITION BY department_id)) as
     avg_dep_salary
FROM
    employees;
```

- PARTITION BY: like GROUP BY, separates rows into partitions
- ORDER BY: order in which rows are processed
- ROWS BETWEEN (start, end): which rows to process
  - *start* PRECEDING AND *end* FOLLOWING
  - UNBOUNDED PRECEDING AND *end* FOLLOWING
  - CURRENT ROW
- LAG(*column*, *offset*): Access previous rows per defined offset value
- LEAD(*column*, *offset*): Access following rows per defined offset value
- RANK(): specify rank each row by a defined column
- DENSE_RANK(): specify a unique rank number, unlike RANK() if we have duplicate values, the duplicate has the same rank but the rank does not give any gap for the values. i.e..



- ROW_NUMBER(): get a unique sequential number for each row even if they have duplicates. i.e..



- CUME_DIST(): finds cumulative distance (0,1]
- PERCENT_RANK(): finds percentile [0,1]

5 Databases and Systems

- Primary Key: field uniquely identifying each row
- Foreign Key: field linking two related tables
- Normalization: process of separating data to prevent redundancy
- Denormalization: optimization technique to keep redudndant data to prevent expensive join operations
- Database view: like a normal table but with no schema. Advantages include: simplification of workflow by aggregating tables, dynamically computed meaning less memory overhead, limited table exposure
- Properties of Distributed Databases

  - CAP Theorem for assessing properties of a distributed db
    Consistency: all clients using db see same data
    Availability: system is always available
    Partition tolerance: system functions even if node communication is lost/delayed
  - ACID framework for measuring correctness and completeness of a relational db transaction
    Atomicity: entire transaction occurs as a whole or it doesn't occur at all
    Consistency: ensures db is consistent before/after a transacation is completed
    Isolation: transaction occurs in isolation so that multiple transactions occur independently without interference
    Durability: once a transaction is completed, the db is properly updated
  - BASE framework for NoSQL databases, similar to ACID
    Basically Available: There will always be a response to any request, but the data may be inconsistent and inaccurate
    Soft State: systems's state may change over time, even without input
    Eventual Consistency: data will converge to a consistent state, no guarantees of when
- Scaling Databases
  - Vertical scaling: add CPU and RAM to existing machines. Easy to administer but expensive as certain machines may be close to their physical limits
  - Horizontal scaling: add more machines (nodes) to the resource pool. Much cheaper and better fault tolerance but difficult to administer (data consistency between nodes)
  - Sharding: Split rows across nodes in a cluster
- Relational Databases: store data in a table-based structure
- NoSQL Databases: store data in forms other than table-based
  - Document databases - allow complex, nested, varied schemas inside of it (i.e. MongoDB, Elasticsearch)
  - Graph databases - data stored by relationships to other data records (i.e. Neo4J)
- MapReduce: parallel big data processing framework
  - 1. Split: splits input data and distributes across nodes
  - 2. Map: take input data and output <key, value> pairs
  - 3. Shuffle: move <key, value> pairs with same key to same node
  - 4. Reduce: aggregate <key, value> pairs into final output
- Spark: another parallel big data processing framework. Faster than MapReduce as it uses RAM for computation enabling faster in-memory performance but higher running costs