**SQL Select Statement**

**SELECT** List the columns that should be returned from the query

**FROM** Indicate the tables from which data will be obtained from

**WHERE** Indicate the conditions to include a tuple in the result

**GROUP BY** Indicate the categorization of tuples

**HAVING** Indicate the conditions to include a category

**ORDER BY** Sorts the result according to specified criteria

**SQL Statements** are not **case sensitive**; HOWEVER, the naming conventions of attributes and tables are **case sensitive** and does not permit ‘- ‘characters

## SELECT

* **UNIQUE**
  + List attributes that comprise a candidate key
* **DISTINCT SELECT DISTINCT** country…
  + To force eliminate any duplicates while search
* **Simple arithmetic operations SELECT** age\*2, year/5, age + 4-2 …
* **AS SELECT** old\_name **AS** new\_name
  + Renaming attribute names
* **AGGREGATE FUNCTIONS**
  + must use **DISTINCT** in addition to aggregate over **sets**
  + ignores **NULL** values.

|  |  |
| --- | --- |
| **AVG()** | Average Value |
| **MIN()** | Minimum Value |
| **MAX()** | Maximum Value |
| **SUM()** | Sum of Values |
| **COUNT()** | Number of values |

## FROM

* Lists the relations involved in the query

|  |  |
| --- | --- |
| Available join types | Join condition |
| * **INNER JOIN** * **LEFT OUTER JOIN** * **NATURAL JOIN** * **RIGHT OUTER JOIN** * **FULL OUTER JOIN** | * **NATURAL** * **ON** <CONDITION> * **USING** <ATTRIBUTE LIST> |

## WHERE

* Specifies conditions that the result must satisfy
* **Comparison operator** =,>,<,>=,<=,!=
* **AND, OR, NOT**
* **BETWEEN … AND … WHERE** mark **BETWEEN** 75 **AND** 100
* **LIKE …** **WHERE** uos\_Code **LIKE** ‘COMP%’
  + Wild cards
    - ‘%’ : Matching **any number of characters**
    - ‘\_’ : Matching **single character**
* **Concatenation** **‘||’** **WHERE** unikey = name **||** number
* **UPPER(), LOWER() WHERE UPPER(**name**) =** ‘JOSH’
* **CHAR\_LENGTH() WHERE CHAR\_LENGTH(**name**)** = 4
* **IS** **(NOT) NULL** **WHERE** NAME **IS NOT NULL**

## ORDER BY “…**ORDER BY** name **DESC**, age **ASC”**

* **ASC**
  + Ascending order (default)
* **DESC**
  + Descending order

# TABLE CREATION AND OPERATIONS

|  |  |  |
| --- | --- | --- |
| Create table **CREATE TABLE** name(  Attribute\_name data\_type constraints  ) | Delete Table **DROP TABLE** name   * Schema and instances are deleted | Change Existing schema **ALTER TABLE** name  **ADD COLUMN**  **ADD CONSTRAINT** |

## PostgreSQL datatypes

|  |  |
| --- | --- |
| SMALLINT  INTEGER  BIGINT | Integer value |
| DECIMAL(p,q)  NUMERIC(p,q) | Fixed-point numbers with precision p and q decimal places |
| FLOAT(p)  REAL  DOUBLE PERCISION | Floating point number with precision p |
| CHAR(q)  VARCHAR(q)  CLOB(q) | Binary string of size r |
| BLOC(r) | Binary string of size r |
| DATE | Date |
| TIME |  |
| TIMESTAMP |  |

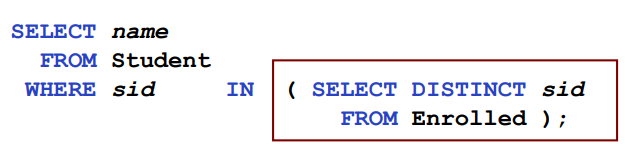
## PostgreSQL constraints

|  |  |  |  |
| --- | --- | --- | --- |
| **NOT NULL** | | | Set it so that no value in a given column can be null |
| **PRIMARY KEY** | **Primay Key (attr1, attr2)** | | Unique, Not null values by default. |
| **FOREIGN KEY** | Attribute\_name data\_type **REFERENCES** Table(attribute) | | Enable a **dependant relation** to refer to it’s **parent relation** |
| **UNIQUE** | **UNIQUE(attr1,attr2)** | | Make it so that the value in the attribute(pair) is unique in the table |
| **CASCADE** | **ON UPDATE**  **ON DELETE** | **CASCADE**  **DEFAULT**  **DELETE** | When foreign key gets changed, the content of the referenced value will be reacting accordingl |

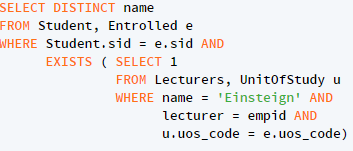
# Instance Modification

|  |  |  |
| --- | --- | --- |
| Insert to table **INSERT INTO** table\_name (list\_attr) **VALUES** (values) | Update **UPDATE** table\_name  **SET** attr\_name = value  **WHERE** condition | Delete **DELETE FROM** table  **WHERE** Condition |
| **INSERT INTO** Student(sid,name) **VALUES** (480222279,”Kim”) | **UPDATE** Student  **SET** sid = 480222279  **WHERE** lname = ‘Kim’ | **DELETE FROM** Student  **WHERE** name = ‘Kim’ |

## Nested Subqueries

When **SELECT-FROM-WHERE** statement is nested within another query

### Noncorrelated subqueries

* Does not **depend** on **data** from the **outer query**
* **Executes once** for the **entire outer query**

### Correlated Subqueries

* Makes use of **data from the** **outer query**
* **Executes once** for **each row** of the **outer query**
* Can use the **EXISTS** operator

## In vs Exists

**IN**

* Compares a **value v** with a **set of values V** and evaluates **TRUE** if **v** is **one** of the **element in V**

**EXISTS**

* Used to check whether the result of a correlated nested query is **EMPTY** or not

## Subquery operators

|  |  |  |
| --- | --- | --- |
| (NOT) EXISTS | Tests whether a set is empty or not |  |
| UNIQUE | Tests whether a subquery has any duplicate tuples |  |
| ALL | Tests whether a predicate is true for the whole set |  |
| SOME | Tests whether some comparison holds for at least one set element |  |