SQL

SQL language statements Groups

- Data Definition Statement (DDS);
- Data Manipulation Statements (DMS);
- Queries; and
- Data Control Statements (DCS)

Data (Table) Definition Statements

Create Table

```
CREATE TABLE cars (
    carID           INT         GENERATED ALWAYS AS IDENTITY                PRIMARY KEY,
    serialNumber    INT         CONSTRAINT notUniqueSerialNumber UNIQUE     NOT NULL,
    manufacturer    VARCHAR(20) NOT NULL,
    model           VARCHAR(20) NOT NULL,
    CONSTRAINT nameOfConstrain CHECK (REGEXP_LIKE(serialNumber, '^[a-zA-Z0-9]+$'))
);
```

> You need at least 1 column

`REGEXP_LIKE` : Regular Expression Like It checks the string is in accordance with the pattern.

Delete Table

```
DROP TABLE cars;
```

Inset Data

```
INSERT INTO cars (serialNumber, manufacturer, model)
VALUES ('12124ABC', 'Jaguar', 'red');
```

> if you want you can remove the columns if you can insert in the rows as they are listed in the Crete Table

Delete data from Table

```
DELETE FROM cars
WHERE model = 'red';
```

> if you don't put `WHERE` all data is removed

Modifying Data on Table

```
UPDATE cars
SET manufacturer = 'Ferrari',
    model  = 'SuperRed'
WHERE serialNumber = '12124ABC';
```

Constrains and Comments

`PRIMARY KEY`:

- each row uniquely identified
- unique and not null
- one primary key `UNIQUE`:
- unique and can be null
- multiple unique can be used `CHECK` condition:
- condition that must be for each row `FOREIGN KEY` theRow `REFERENCES` tableName(referencedColumn):
- the referencedColumn should be `UNIQUE` or `PRIMARY KEY` `DEFAULT` value:
- If nothing is inserted the `value` is used `NOT NULL`:
- Ensures that null values cant be entered in this column

Giving names to constrains:

```
CONSTRAINT nameOfConstrain conditionHere;
```

Comments in SQL:

```
-- This is a single-line comment in SQL.
/* This is a multi-line comment in SQL.
You can add comments across multiple lines
within the comment block. */
```

Queries

Querying Generally:

```
SELECT columnNames /*Result Table columns*/
FROM tableNames    /*Tables from witch the resutls are selected*/
WHERE conditons    /*Which rows to retrive*/
GROUP BY criteria
ORDER BY criteria;
```

> `WHERE`, `GROUP BY`, `ORDER BY` are optional

Projection(SELECT):

```
SELECT serialNumber, model FROM cars; /*Select those columns*/
SELECT * FROM cars;                   /*Select all of the columns*/
SELECT 100*CARID, model FROM cars;    /*Using arithmetic expression*/
```

> The `SELECT` can result in identical lines they can be removed using the `DISTINCT` Example many cars can have the same manufacturer

```
SELECT DISTINCT manufacturer FROM cars;
```

`As` keyword:

```
SELECT 100*CARID AS CARID00, model AS Color FROM cars;    /*Naming the rows of the Result Table*/
```

Question: Why is `AS` not working ins SQL developer?

Restrictions(WHERE)

```
WHERE predicate
```

Predicate can have:

- `literals`:
  - numbers
  - string
  - dates
  - column names
- `operators`:
  - for numbers:
    - arithmetic operators (+, -, *, /)
    - arithmetic functions `WHERE ABS(balance) < 1000`
  - for strings:
    - `SUBSTR()`: `WHERE SUBSTR(product_name, 1, 3) = 'ABC'`
    - `INSTR()`: `WHERE INSTR(description, 'important') > 0`
      - It returns the position (index) of the first occurrence of a specified substring within a given string.
      - If the substring is not found, it returns 0.
    - `UPPER()`: `WHERE UPPER(last_name) = 'SMITH'`
    - `LOWER()`: `WHERE LOWER(city) = 'new york'`
    - `SOUNDEX()`: `WHERE SOUNDEX(name) = SOUNDEX('John')`
      - It converts a string into a phonetic code (a four-character code) based on its pronunciation.
      - Find words that sound similar even if they are spelled differently.
  - for dates:
    - - - `WHERE order_date + 7 <= NOW()`
    - conversions `WHERE order_date + 7 <= NOW()`
  - for evaluating:
    - <, <=, =, !=, >=, >;
    - BEEWEEN ... AND ... `WHERE ABS(balance) < 1000`
    - IS NULL, IS NOT NULL;
    - IN set;
    - LIKE pattern `WHERE email LIKE '%@gmail.com'`
      - % every string
      - _ only on char
    - AND OR NOT
- `functions`:
  - `NVL(columnName,value)`: if NULL you just take the value
  - `DECODE(expression, search_value, result1, search_value2, result2, ..., default_result)`:
    - for each `search_value` $\in$ `expression` you assign a new value
    - if there is a value in expression no covered than you use the defaut
    - Find the example here
- `sets`:
  - (10,20,30)
  - `` `WHERE department IN ('Sales','Marketing','Support')`` ``
- `Nested Queries`:
  - WHERE (SELECT ... FROM ...)

Join

Inner Join

```
SELECT A.column1 , B.*  /*Projection of the columns*/
FROM A, B               /*The tables taking part on join*/
WHERE A.ID = B.ID;      /*The inner Join predicate*/
```

> If their is no matching than the joined table doesn't contain the result

Outer Join

```
SELECT A.column1 , B.*
FROM  cars A, tractors B   /*Giving tables local names*/
WHERE A.ID = B.ID(+);      /*The outer Join predicate*/
```

> You get all the entries in A and there matching correspondence in B In another words you are adding B to A Notice the local names.

Aggregate Functions

f: `returnTable.column` -> `oneSingleValue`

- `AVG()`: average value for numbers
- `SUM()`: total of numeric column
- `COUNT()`: number of rows in column or COUNT(*) for the return table
- `MAX()`: select the highest value for column
- `MIN()`: select the lowest value for column

> IF result contains `aggregate functoins` and `constants` than is okay but if a column with many rows is introduced than the programmer should group

Bad:

```
SELECT COUNT(*), descrip FROM product;
----------------------------------------
SELECT startdate, AVG(stdprice) FROM price
WHERE startdate = '01-jan-94';
```

Good:

```
SELECT startdate, AVG(stdprice) FROM price
WHERE startdate = '01-jan-94'
GROUP BY startdate
```

Nested Queries (Subquery)

Generally

```
SELECT ... FROM ...
WHERE ... (
    SELECT ... FROM ...
    WHERE ...
);
```

or

```
SELECT ...
FROM ... (
    SELECT ... FROM ...
    WHERE ...
);
WHERE ...
```

Questions:

```
SELECT prodid, descrip FROM product
WHERE prodid IN (
    SELECT prodid FROM price
    WHERE startdate >= '01-jan-94'
)
```

The `prodid` is a row and not a set when I do the query i get a repeating values:

prodid
1
2
3
2

Grouping

Effective when you have aggregate functions. You can apply those functions to specific group. Lets suppose you have all the students and they are in different years. You want to find the average grade every year. So what you do is

```
SELECT yearOfStudies, AVG(grade)
FROM students
GROUP BY yearOfStudies;
```

> you can only have one column and the aggregate functions applied to it

If you only want first and second years than:

```
SELECT yearOfStudies As year, AVG(grade)
FROM students
GROUP BY yearOfStudies
HAVING year<=2;
```

> If you can use `WHERE` but sometimes you may have to use `HAVING`. Here we can use `WHERE` because the `year<=2` does not have condition related to an aggregation. So we use `HAVING` when we have a condition related to aggregation

Better example: Filters years where the average is greater than 3

```
SELECT yearOfStudies As year, AVG(grade) AS avg
FROM students
GROUP BY yearOfStudies
HAVING avg>3;
```

Ordering

```
SELECT ... FROM ...
ORDER BY column order, column order...;
```

`Order` can be:

- `ASC`: default
- `DESC`

Set Operations

> The return Table from the `SELECT` can be considered as a set. Therefor the set operations can only be between 2 `SELECT` statements.

- `UNION`
- `INTERSECT`
- `MINUS`
- `IN`

Hierarchical relationship query

```
SELECT name, parentID, childID
FROM tableName
CONNECT BY PRIOR parentID = childID  /*A recurisive call*/
START WITH name='SuperParent';       /*If you use the super parent it lists everything */
```

Footer

Decode:

```
SELECT
    DECODE(STUDENT_CODE, 'S001', 'Beni', 'S002', 'Klevis', 'S003', 'Sara', 'NoOne') AS code,
    AVG(BEGINNING_YEAR_OF_STUDIES) AS avg_beginning_year
FROM
    STUDENTS
GROUP BY
    STUDENT_CODE
ORDER BY
    avg_beginning_year DESC
```