**SQL**

il client richiede un ‘Qualcosa’ dal database server. Il DS invierà delle risposte.  
Il database server è in attesa di istruzione (date dal client).  
Per inviare comandi al database ci sono diversi COMANDI.  
Le risposte date dal database si chiamano RISPOSTE.  
La rete è uno dei modi con cui si realizza la modalità dei server.  
La rete svolge una funzione principale. La comunicazione può avvenire su più coputer.  
Necessita quindi una **connessione**!  
Per effettuare una connessione si necessita di

* Porta SP
* Indirizzo IP.

**LA CONNESSIONE**

Quando un computer si connette alla rete.  
La porta standard del processo oracle è per default ‘Installata’ dall’installer di default.  
Se non ci si deve allacciare alla rete per comunicare con il pc, ci si rivolgerà al Local Host/127001.  
127001 identifica la propria rete!  
Quello che facciamo dal developer al server, potremmo farlo anche da un computer dall’altra parte del mondo.  
Il db server (ci può arrivare chiunque) dovrebbe avere un sistema di sicurezza.

**ORACLE**

Per lavorare con un database server necessita una tipologia di connessione permanente.

Internet usa un protocollo diverso: http (protocollo senza connessione).

Il server quando parte, oltre a mettere a disposizione la porta 1521, si prende un pezzo di disco e gestisce una parte di spazio per archiviare i dati.  
Quando un client ha bisogno di qualcosa lo chiede al client.  
Il client ‘Parla ‘solo con il server, non sa fare altro!

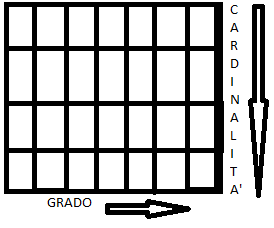
**IL SERVER**

Il server è un Relational Database Management System.

L’obiettivo di un database server è quello di memorizzare e rileggere dei dati in modo strutturato.   
il modo strutturato è il relazionale che si va ad imparare.

Il database permette di memorizzare e recuperare dati preinseriti nella ***FONTE*** (tabella).  
**STRUTTURATO**: le informazioni sono strutturate all’interno.   
Si definiscono regole sotto le quali il dato deve stare.  
Una volta definite le regole, posso memorizzare il dato che segue le regole create ed estrarlo per metterlo via. (ispezionarlo in modo più preciso).

**Quali sono le regole per strutturare i nostri dati?**

****

C’è una modalità con cui i dati devono essere memorizzati. (**TABELLA**)

La tabella è una porzione di disco che contiene dati.  
La tabella di un database relazionale risponde a delle regole. E’ composta da una griglia.  
Le colonne saranno chiamate FIELDS, le righe RECORD. Una tabella è un contenitore di RECORD.

Nella tabella non c’è modo di inserire un record che la tabella dice come è fatto il record.  
Le regole quindi sono molto rigide!

Il numero di elementi (RECORD) che la tabella contiene si chiama **CARDINALITA’** DELLA TABELLA.  
Il grado è il numero di campi contenuti nella tabella.  
  
In un database ci sono:

1. La struttura del dato
2. Il dato.

Prima di poter salvare un record devo aver definito una tabella!

Nel RDBSM esistono diversi insiemi di comandi:

1. **Data Definition Language**: creano, modificano, eliminano strutture;
2. **Data Manipulation Language**: creano, modificano, eliminano i dati.

Le tabelle possono essere divise secondo l’uso che si fa!  
Esse vanno a identificare spazi presso cui creiamo lo **SCHEMAS**.  
Schema: raggruppamento logico di oggetti nel database.  
Perimetro dentro al quale si lavora.

Lo schema si può creare in ogni database.

In Oracle, schema, è sinonimo di ‘User’. Schema, quindi è uguale ad utente!  
Quando in Oracle creo un utente, tutte le tabelle finiranno nel suo schema

Il DBA è il potere più grande che si può concedere ad un utente in SQL.

Quando invio un comando si riferirà al default schema.

**REGOLE TABELLE**

In uno schema non ci possono essere più oggetti con lo stesso nome.  
ha una lunghezza massima di 32 caratteri.

Non contiene spazi, accenti (non discostarsi dagli standard).  
In SQL il “ \_ “ svolge la funzionalità del camelCase di JAVA!

In SQL i tipi carattere hanno il problema della codifica caratteri.  
Devo dividere quali sono i byte salvati sul disco (unicode, ecc).   
QUANTI BYTE PER CIASCUN CARATTERE UTILIZZARE?

Se1 byte per carattere: **VARCHAR**

2 byte per carattere : **VARCHAR2** (sono più portabili)

CHAR : LUNGHEZZA FISSA

VAR CHAR: LUNGHEZZA VARIABILE

Il database è stato sviluppato dall’algebra funzionale. L’algebra rappresenta i nostri dati come insiemi (teoria formale-matematica). Il criterio matematico definisce alcune regole che noi implementeremo.  
1. All’interno di una tabella il database relazionale deve distinguere in modo univoco gli elementi della tabella.

**La Primary Key [PK]** è un campo (una colonna sola) i cui valori non si ripetono mai all’interno della tabella.

I contatori sono l’esempio più immediato.

La PK identifica un elemento all’interno di una tabella. In ogni tabella devo definire una o più campi utilizzati come chiavi primarie!

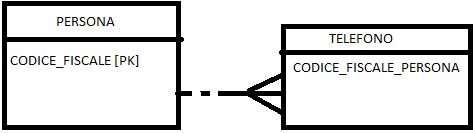
* **Chiave** **naturale** (regola esistente nella realtà che tramuto nel programma. Ad esempio il codice fiscale).
* **Chiave** **surrogate**= strumenti. “Siccome non trovo nessun esempio pratico, concreto nella realtà, lo invento io.
* **Chiave** **esterna**: chiave ereditata da una tabella creata in precedenza.

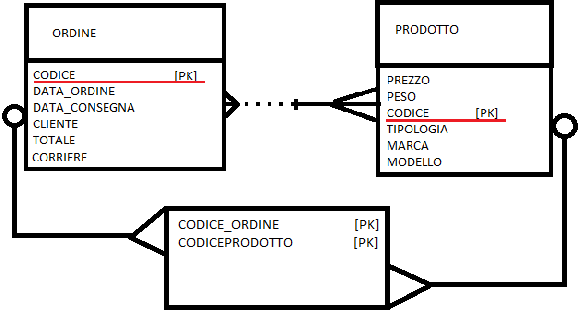
------------------------------------------------------------- GIORNO 2-------------------------------------------------------------------

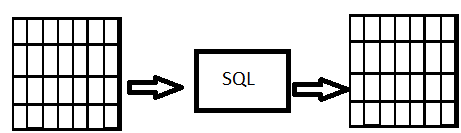
**Il funzionamento del database relazionale**

Ogni tabelle DEVE avere una PK, poiché è lo strumento con il quale utilizziamo le informazioni che inseriamo.  
La PK quindi ha una funzione identificatrice.

Associazione 1/n (uno a molti) :



Associazione N/n (“N” a “Molti”)

****Come estrarre dei dati dalla **Base Dati.**

1. Interrogare la base dati utilizzando il linguaggio SQL. Linguaggio formale (per pc), elaborato e processato da un IDSM. Vi è il medesimo modello algebrico che da origine alla forma del sistema. Esso ci consente di applicarlo e far funzionare

**DATA MANIPULATION LANGUAGE**

1. Creare un foglio di lavoro dove si scriveranno programmi SQL.  
   Il linguaggio SQL è un **linguaggio funzionale**.  
   Non ci sono cicli, ma c’è una funzione di trasferimento (trasforma un insieme in un altro insieme).

Esso prende degli elementi tabulari (matrici che hanno già regole date in precedenza), e li trasforma in una nuova matrice.

**LE OPERAZIONI RELAZIONALI**

1. **LA PROIEZIONE**

Essa serve per modificare il grado (*numero di colonne*) del risultato. Permette quindi di aggiungere o rimuovere colonne.

Si possono proiettare:

* Le costanti
* Gli identificatori
* Le espressioni ( prezzo / 2);

**La funzione “select \*”(star)**

Recupera righe dal database e consente la selezione di una o più righe o colonne da una o più tabelle in SQL Server. Inoltre, la “select \*” non mi vieta di usare espressioni all’interno. Inoltre prende sempre la cardinalità della tabella presa!

\*. Dual + una tabella che mi permette di fare conti e calcoli.

La sintassi è la seguente.

Agisco sulla cardinalità.

SELECT column1, column2, ...  
(es. data ora corrente su Oracle 🡺

Select sysdate  
fine dual

1. **LA SELEZIONE**Essa mi permette di selezionare dati dal database decidendo inoltre una ‘Condizione’

**Where –condizione booleana-**

SELECT column1, column2, ...  
FROM table\_name  
WHERE – *boolean* condition – ;

Nella condizione booleana sono permessi:

1. = | Uguaglianza
2. < > | Disuguaglianza

Agisco sul grado

1. < | > | <= | >= |
2. Valori Letterali;
3. Funzioni;

La condizione booleana viene valutata riga per riga e stamperà tutte le righe che risultano true.

Es.  
Where prezzo >50 🡪 stamperà le sole righe dove la condizione è true.

1. **L’ORDINAMENTO**

Mi permette di visualizzare in maniera ordinata (ascendente o discendente) secondo un criterio che si decide una riga!  
Per default ordina in modo ascendente.

La sintassi è la seguente.  
  
SELECT column1, column2 …..  
FROM table nome  
ORDER BY column1, column2, … ASC|DESC

1. **LA CONGIUNZIONE**

La clausola di congiunzione è utilizzata per combinare righe da righe o tabelle basate su colonne combinate tra di loro.

La sintassi è la seguente.

SELECT Nomi Colonna(e)  
FROM Tabella1 INNER JOIN Tabella2  
ON campo1 = campo2;

Se è presente un’omonimia, si deve provvedere a risolvere il ‘Problema’

* ***INNER JOIN:*** trova solo le corrispondenze correlate delle due tabelle;
* ***OUTER JOIN:*** verifica l’uguaglianza o la nullità di due termini.

------------------------------------------------------------- GIORNO 2-------------------------------------------------------------------

Database: composti da tabelle.  
Tabelle: 1) composte da colonne (Field[s]); 2) Composte da Row;

Sql viene usato poiché mantiene in maniera sicura l’integrità.

**DATA INTEGRITY:**

* **Entity**: non ci sono duplicati (righe identiche ILLEGAL!)
* **Domain**: ci garantisce che i dati abbiano sempre senso in base alla tipologia di colonne che abbiamo scelto. (No data type error)
* **Referential**: Nel caso di tabelle che dipendono da altre tabelle non possiamo cancellare righe che dipendono dalle stesse
* **User-defined**: Ciò che viene stabilito dall’utente.

ESEMPI di QUERY.  
  
***SELECT*** [campi e funzioni]  
***FROM*** [tabelle  
***WHERE*** [condizioni booleana]  
***GROUP*** raggruppare risultati in base a delle condizioni  
***HAVING***

**DATA TYPE**

* **Numerici:**  big int (+- 9\*1016, int, small int, tiny int, Decimal, Money, SmallMoney, float, real.
* **Date**: datetime( date + time) (parte dal 1/01/1753), smalldatetime(parte dal 12:00 1/01/1970),date, time.
* **Char** : char (occupa 8000 spazi di caratter), varchar(occupa 8000 caratteri ma possiamo definirne il numero con n.char), text (salva solo un carattere).
* **Unicode Char**: nchar, nvarchar, ntext
* **Misc**: timestamp, image, blob ( si salvano immagini e file), bin, textvarbin (…) ( riservano tot byte in base a ciò che gli si passa).

**KEYWORDS**

**ALL**: richiama tutti i record;   
**AND**/**OR**: concatena condizioni;  
**ANY**: Un qualsiasi campo; (Where Age > ANY ( 13, 25, 37) (prende il valore minimo (>13))  
**BEETWEEN**: Si usa per definire un limite nella condizione ( es Salary beetweem 5 and 6 );  
**EXIST**: servono per le subquery  
**IN**: servono per le subquery; (Es. WHERE Age IN (13, 25, 37) = Lista solo le persone di anni 13  
**ALL**: Serve a (esempio: SELECT … || FROM emp || WHERE Age > ALL Prende il numero più alto (37));   
**LIKE**: Si usa per selezionare tutti gli oggetti che iniziano per un valore che definiamo noi. (es. Nome LIKE “T%”, dove %= 0+, \_= 1 nomi che iniziano per “T”);  
**NOT**: serve per ribaltare la condizione;  
**IS** **NULL/ NOT NULL: /**escludono tutte quelle che non sono null  
**UNIQUE**: serve a forzare che una data colonna non sia mai ripetuta in più di una riga.

**CREATE A DATABASE**

1 DROP  
2 USE NDB  
3 CREATE TABLE (creazione di tabella) NAMETABLE ( column type [vincoli], column2 type [vincoli], primary key(column1);

**ESEMPIO**

CREATE TABLE Customers (  
Id—(numero)-- INT NOT NULL ===== crea un campo di tipo intero non nul  
 AUTOINCREMENTAL, -- dato un valore, aggiungerà 1 all’ultima riga  
Nome VARCHAR(20) NOT NULL CHECK>18,   
Age INT NOT NULL,  
Address CHAR(50),  
Salary Decimal (18,2),  
PRIMARY KEY (id));

**ESEMPIO 2**

CREATE TABE SCustomer

SELECT Id, Age

FROM Customer  
WHERE Age BEETWEEN 18 and 30;

Prende dei dati strutturati da una tabella e va a creare una tabella che va a riprendersi strutture ed informazioni da strutture precedentemente create.

**DROP TABLE:** E’ usato per rimuovere una tabella esistente in un dato database. **TRUNCATE TABLE:**  Rimuove tutte le righe da una tabella o le partizioni specificate di una tabella, senza registrare le eliminazioni delle singole. TRUNCATE TABLE è simile all’istruzione DELETE senza clausola WHERE. L’istruzione TRUNCATE TABLE è tuttavia più rapida e utilizza un numero minore di risorse di sistema.

Quando noi dobbiamo inserire dati

INSERT INT() Customer  
(Nome, Age, Address; Salary)  
(col1, col2 colN)  
VALUES (d1,d2,dN);  
(“Piero”, 35, 50);  
INsERT INT NomeTabella  
(col1, col2, colN)

SELECT col1, col2, colN  
FROM ATB  
WHERE […]

UPDATE Customer  
SELECT Nome=”Marco”;  
 Salary= 1500  
WHERE Id>35

DELETE FROM Customer  
WHERE Id=18 – Cancella una o più righe!

**FUNZIONI PARTICOLARI**

1. TOP

SELECT TOP 5 \*  
FROM Customer  
ORDER BT DESC Salary – Mette nella stampa i primi 5 che prendono di più!  
+ Top fornisce anche la percentuale

2) ORDER BY  
SELECT   
FROM Customer  
ORDER BY DESC Salary   
LIMIT 10 – Tronca il risultato

2) SELECT DISTINCT

SELECT DISTINCT   
The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

The SELECT DISTINCT statement is used to return only distinct (different) values.

Salary   
FROM Customer – Tronca il risultato -- Evita i duplicati

I VINCOLI

**Costraints**

*NOT NULL* 🡺 va a garantire che ci sia un dato.   
*DEFAULT* val 🡺 Inserisce un valore impostato  
*UNIQUE*🡺 Lavora come la chiave primaria (evita la duplicazione di uno stesso dato nello stesso campo  
*PRIMARY*/*FOREIGN* *KEY*🡺 creano indici sul campo (pk) e ci permette di avere combinazioni non ripetute. Vanno a riferirsi a chiavi di un'altra tabella  
*CHECK* 🡺 Controlla i vincoli determinati dallo user ed impone la condizione  
*INDEX* 🡺 a) FAST SEARCH. B) SLOW/INSERT/UPDATE/ DELETE   
E’ simile ad un sommario di un libro. Si crea su uno dei campi presenti. Quando si usa su un campo, la ricerca sarà molto efficiente

Per liberarsi di un vincolo:

ALTER TABLE   
**DROP COSTRAINT**FOREIGN KEY

Per creare un INDEX:

CREATE INDEX nome.index  
ON NomeTabella

ALTER TABLE DROP INDEX

Nome\_index

LA UNION

SELECT Name, Salary  
From Customer  
Where Name LIKE “8%”  
UNION ALL – Si utilizza ALL poiché si vuole RIPETERE i duplicati  
SELECT Name, Salary  
FROM Customer  
WHERE Name LIKE “P%”;

SELECT Name, Salary  
From Customer  
Where Name LIKE “8%”  
EXCEPT – ”Tutto tranne”  
SELECT Name, Salary  
FROM Customer  
WHERE Name LIKE “P%”;

GLI ALIAS

Possono essere utilizzati sia su campi che su tabelle.

Sono dei nomi temporanei assegnati a delle operazioni.  
   
AS

SELECT C.Name AS N   
FROM Customer AS C  
WHERE N LIKE AS “T%”;

get.name  
  
CREATE VIEW get.name AS  
SELECT Name  
FROM Customer  
  
SELECT \*  
FROM get.name  
WHERE Name LIKE “T%”

DROP VIEW ger.name

LE TRANSACTION

ACID = Atomicity Consistency Integrity, Durability

Una transazione è un insieme di query separati da punti e virgola.  
Viene terminata dal commit che esegue tutto ciò che sta sopra.  
Fornisce i punti di ripristino e i rollback.

**Atomicity** = Garanzia che o viene eseguita TUTTAla query oppure NESSUNA  
**Consistency**=  
**Integrity**:   
**Durability**= I nostri dati restano al sicuro

SAVEPOINT nome.point (salva la parte attuale del database)  
ROLLBACK nome.point (riporta alla fase di pre-consistenza il query)  
SET TRANSACTION nome.point = ???

**LE FUNZIONI: LE DATE**  
  
DATE\_ADD(‘2018-02-21’, INTERVAL 31.DAY) 🡺 Crea una tabella 2018-03-22. Somma varie unità di tempo

DATE\_ADD(CURDATE()) 🡺 Ci da la data corrente.

CURTIME() 🡺 Ci da l’orario  
DATEDIFF(date1, date2) => Differenza tra le due date  
DATE\_FORMAT(‘2018-02-21 12:30’, ‘%H%n’)🡺 Rappresenta la formattazione della data

CREATE TEMPORARY TABLE nome.th.temp  
SELECT camp1, camp3  
FROM table.name  
WHERE condizione

**LE SUB-QUERY**

SELECT \*  
FROM Customer  
WHERE Id IN (

SELECT Id  
FROM Customer  
WHERE Salary > 500

INSERT INT() temp.table

SELECT \*  
FROM Customer  
WHERE Id IN (

SELECT Id  
FROM   
WHERE Age>30

UPDATE Customer

SELECT Salary = Salary \* 0,25  
WHERE Age IN (  
SELECT Age  
FROM Customer  
WHERE Age>30)

Aggiorna il salario e gli toglie il 25%

String nomeUt = “Piero”

SELECT \*   
FROM User  
WHERE Nick LIKE ‘Piero’;  
  
nome Ut = “Piero/”; DROP DATABASE MAIN;

SELECT \*  
FROM User  
WHERE “Piero”, DROP DATABASE MAIN;

LE FUNZIONI

**NVL**: Ci ‘Ritorna’ una stringa dal risultato di una query in risposta alla condizione null.

TO\_CHAR:

**SELECT TRUNCATE** (Salary \* 1.034467, 2) 🡺 ci tronca i decimali a due!)

La differenza di anni tra due date.

SELECT to\_char (CURDATE, ‘YYYY’) -   
 to\_char(Birthdate, ‘YYYY’)

FROM EMPLOYEES

SELECT SUBSTR(Name, 2, 10) 🡪 numero di partenza 2, numero di caratteri 10;

SELECT ColumnName|| ‘,’ || ColumnName2 ci permette di concatenare sia colonne sia stringhe

LE FUNZIONI COLONNA

SELECT SUM(Salary) SUM.  
 MIN(Salary) MIN,

ATTENZIONE AI VALORI NULL!!!!

MAX(Salary) MAX,

AVG(Salary) AVG,

COUNT(\*) COUNT,

COUNT(DISTINCT Name) DCOUNT 🡺 dobbiamo inserire un dato che abbia un senso

**GROUP BY**

SELECT (2)SUM(Salary)

FROM Employees

(1)GROUP BY WorkDepartments

Having (3)SUM(Salary) > 50000