Breve Introduzione a SQL

Claudio Rocchini
Istituto Geografico Militare

L'interfaccia: introduzione

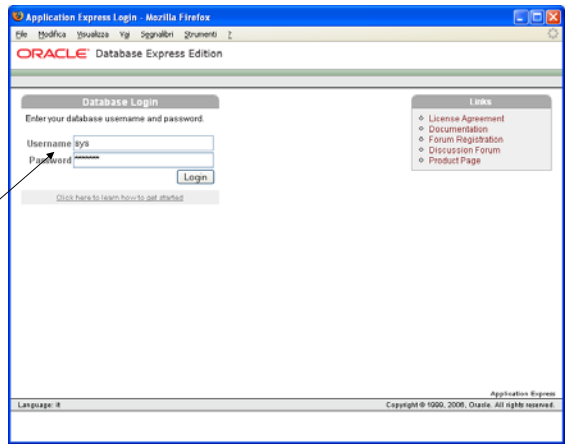
- Per quanto riguarda l'interfaccia di Oracle, utilizzeremo principalmente la versione Express, molto semplice da installare.
- Faremo però anche un cenno di utilizzo dell'interfaccia SQL*PLUS, e dell'importatore di dati DOS.
- Nota: tutto il software Oracle è scaricabile liberamente.

Prima fase

- Collegamento a Oracle Express
- Login con utente sys
- Creazione utente di lavoro
- Connessione con utente di lavoro

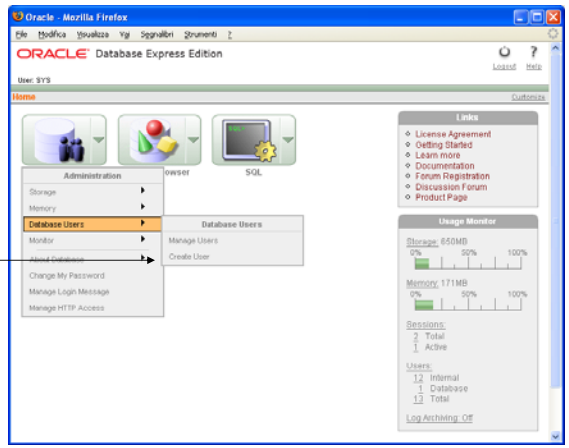
Start dell'interfaccia

- Dal menù start > Oracle , selezionare "Vai alla home page del database".
- Digitare l'utente SYS, con la password selezionata in fase di installazione.



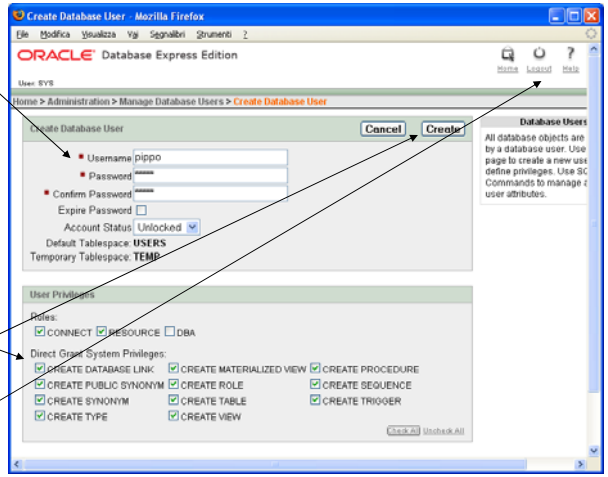
Menù utente

- Selezionare il menù Administrator, Database Users, Add User



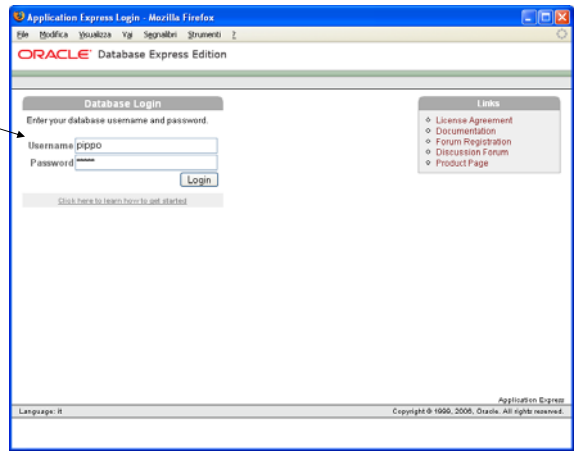
Creazione Utente

- Digitare come nome “pippo”
- scegliere la password
- Ceccare tutti i flags (escluso DBA = amministratore)
- Premere il bottone “Create”
- Premere logout



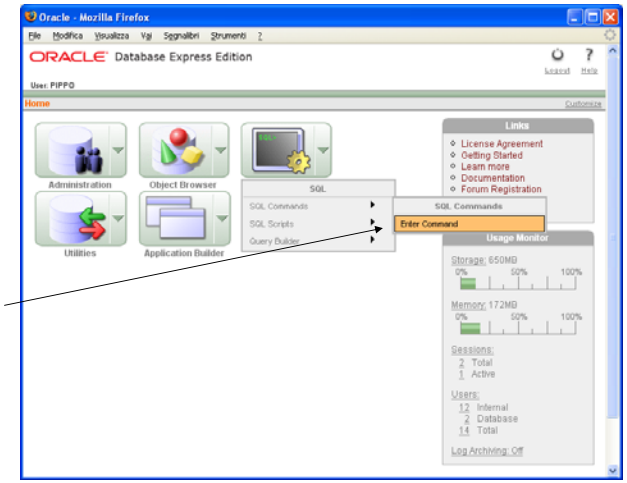
Login con utente di lavoro

- Ci riconnettiamo con l’utente pippo e la password scelta.
- L’utente sys ha delle limitazioni, per cui non può essere utilizzato.



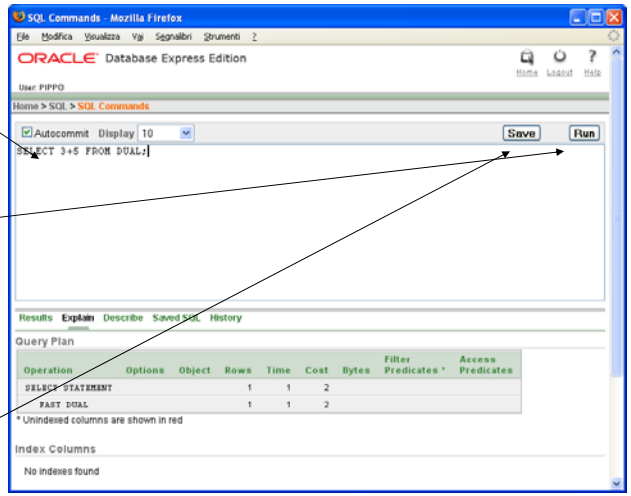
Finestra SQL

- Selezioanare: SQL > SQL commands > Enter Command
- Entreremo nella finestra SQL



La finestra SQL

- In questa finestra scriveremo i comandi SQL
- Dopodichè premeremo il bottone “run”
- I risultati della query appariranno in basso
- E’ anche possibile salvare una query per poi recuperarla.



Le gioie dell’informatica

- L’interfaccia SQL è veramente intelligente:
- Se fate il gravissimo ed imperdonabile errore di scrivere un solo spazio in più all’inizio della query, Oracle si bloccherà per errore tremendo e si rifiuterà di eseguire qualsiasi query SQL, anche se scritta in modo perfetto.
- Ricordatevi di controllare che non ci siano spazi all’inizio della query.

Introduzione

- SQL = Structured Query Language.
- SQL è un linguaggio testuale standard per la manipolazione di basi di dati.
- Standard: comune a tutti i DBMS che lo supportano (Access, Oracle, MySql, PostgreSQL, Microsoft SQL Server, etc.)
- E’ funzionale: un solo costrutto esegue l’operazione specificata, non imperativo (variabili ed elenco di comandi).

Indice degli argomenti

- Elementi di Base (numeri e parole)
- Definizione dei dati (tipi di dato, schemi, colonne e tabelle)
- Inserimento/Modifica dei Dati
- Interrogazione dei dati
- Elementi avanzati: indici, chiavi, relazioni, check e triggers

Maiuscole-Minuscole

- SQL è di solito “case-insensitive”, vale a dire che non si distingue le minuscole dalle maiuscole.
- Tutti i comandi SQL da digitare sono scritti con il **`font Courier`**
- Nei nostri esempi, per chiarezza, scriveremo sempre i comandi di SQL in maiuscolo, mentre scriveremo in minuscolo i valori ed i nomi definiti dall’utente.
- Se si vuole specificare un nome (di tabella o di colonna) in maiuscolo/minuscolo

Pre Introduzione a SELECT

- Il comando fondamentale di SQL è SELECT e verrà spiegato in dettaglio più avanti.
- Dobbiamo introdurlo per effettuare alcune prove sui dati.
- La sintassi minima di SELECT è:

```
SELECT {valori} FROM {tabella};
```

- Siccome specificare una tabella è obbligatorio, nel caso in cui non sia necessario riferirsi ad una tabella, ORACLE ci mette a disposizione una tabella fittizia che si chiama DUAL.

Valori numerici letterali

- Nella vostra finestra SQL, scrivere il comando:

```
SELECT 42 FROM DUAL;
```

- Il risultato sarà 42. La query che abbiamo scritto richiede al sistema il numero 42, la tabella DUAL viene specificata in modo fittizio (dato che è obbligatorio specificare una tabella).
- In realtà nel risultato ci sono due numeri 42: uno è il valore, mentre l’altro è il nome della colonna. Il risultato di una SELECT è sempre una tabella (anche di una sola riga e una sola colonna come in questo caso).
- Per rendere più il risultato provate a scrivere il comando:

```
SELECT 42 AS valore FROM DUAL;
```

Valori numerici letterali (2)

- Una query più interessante può essere:

```
SELECT 23*4 FROM DUAL;
```

- Oppure per i matematici:

```
SELECT cos(3.1415926) FROM DUAL;
```

- Abbiamo visto come rappresentare valori numerici e come si possano calcolare espressioni e funzioni.

Valori letterali testo

- Vediamo adesso come si lavora con i testi (tipo di dato stringhe di caratteri).
- Provate il seguente comando:
  `SELECT 'Buongiorno' FROM DUAL;`
- Le parole letterali vanno sempre racchiuse fra apici. Questo server per distinguerle dai nomi di colonna.
- Se scrivo infatti:
  `SELECT Buongiorno FROM DUAL;`
- Ottengo un errore, dato che la tabella DUAL non possiede la colonna Buongiorno.
- Se devo inserire un apice nella stringa, lo devo scrivere due volte, es.:
  `SELECT 'L''area dell''edificio'' FROM DUAL;`

Nota su PowerPoint

- Questo documento è scritto in Power point
- Tutti gli esempi sulle stringhe prevedono la scrittura del carattere ‘ (apicetto singolo), che è quello standard della tastiera internazionale.
- PowerPoint si ostina a convertire questo carattere in apicetto angolato o peggio rovesciato ‘’
- Si ricorda invece che SQL accetta solo l’apicetto standard

Espressioni con le parole

- Anche con il tipo stringa si possono scrivere delle espressioni, es.:
  `SELECT LENGTH('casa') FROM DUAL;`
- Oppure (il simbolo || corrisponde al simbolo + per i numeri e concatena due parole):
  `SELECT 'pesce' || 'cane' FROM DUAL;`
- Esistono anche delle funzioni che convertono un tipo in un altro (attenzione ‘1984’ scritto fra apicetti è una parola non un numero!):
  `SELECT TO_NUMBER('1984') FROM DUAL;`

Tipi di dato

- I dati memorizzati nelle tabelle di un database appartengono ad un tipo.
- Il concetto di tipo di dato è un concetto base dell’informatica.
- Quando definite un attributo di una feature class di Arcgis, dovete sempre specificare il tipo di dato associato.
- Tipi di nato sono: numeri interi, numeri con la virgola, parole (stringhe di caratteri), ore e date, valori di verità (vero o falso), BLOB (dati binari generici).

Tipi di dato di Oracle

- In Oracle ogni tipo di dato ha un nome, che andrà specificato nel comando di creazione di una tabella.
- I principali tipi sono:
  - **NUMBER(*PRECISIONE,SCALE*)** : numeri
  - **VARCHAR2(*MAX_DIM*):** stringhe
  - **DATE** : data
- Spesso i tipi hanno dei parametri numerici, ad esempio il tipo stringa ha bisogno della definizione del massimo numero di caratteri memorizzabili.

Altri tipi di dato

- I tipi di dato di base sono molti, non solo:
- Nei sistemi moderni (ad oggetti), è anche possibile definire nuovi tipi di dato, a seconda delle esigenze. Questi tipo possono essere anche complessi, cioè essere definiti in forma di oggetti.
- Gli oggetti hanno proprietà e metodi con cui si può interagire.
- Vedremo che il sistema spaziale di Oracle definisce il tipo **SDO_GEOMETRY**, che è un oggetto rappresentante la forma geometrica e il sistema di riferimento di un'entità cartografica.

Definizione dei Dati

- Vediamo adesso la serie di comandi che permette di definire la struttura dei dati (vale a dire delle tabelle).
- I comandi sono:
  - **CREATE TABLE** : crea una tabella
  - **DROP TABLE**: distrugge una tabella
  - **ALTER TABLE**: modifica una tabella
  - **DESCR**: analizza la struttura di una tabella

Creazione di un tabella

- Creiamo adesso la nostra prima tabella:
- Il comando di creazione di una tabella ha la seguente struttura:

```
CREATE TABLE nome_tabella
(
   nome_colonna1 TIPO1,
   nome_colonna2 TIPO2,
   …
);
```

- All'interno delle parentesi tonde bisogna specificare la lista delle colonne della tabella, separate da virgola. Le colonne sono specificate dal loro nome e dal tipo (Attenzione! Tutti i nomi di colonna e tabella devono essere un'unica parola senza spazi: al massimo si può usare la barra di sottolineato _ . Non c'e' differenza fra maiuscole e minuscole).

Esempio di creazione tabella

- Provate a scrivere adesso:

```
CREATE TABLE studenti
(
  nome VARCHAR2 (128),
  eta NUMBER(3),
  codice_corso NUMBER(6)
);
```

- Questo comando creerà la nostra tabella studenti, con tre colonne. Notate la specifica della massima lunghezza (128) e le 2 virgole che separano le colonne.
- Nota: la formattazione (i ritorni a capo e gli spazi) non conta nulla. Scriveremo i comandi in una certo modo solo per renderli più chiari. Potevamo scrivere anche:

```
CREATE TABLE studenti(nome VARCHAR2 (128), eta NUMBER(3),
  codice_corso NUMBER(6));
```

Analisi di una tabella

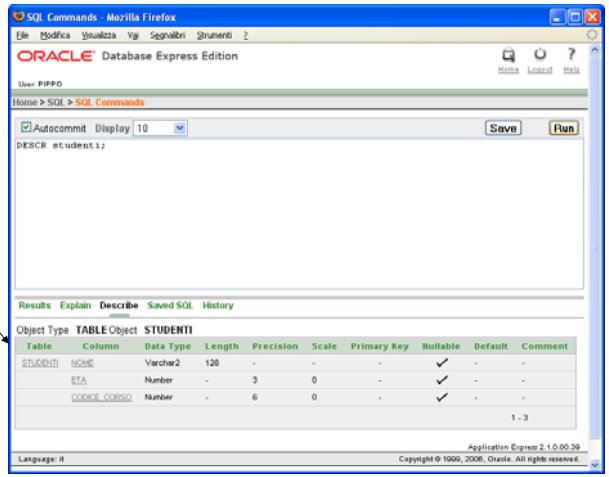
- Una volta che abbiamo creato una tabella, ne possiamo analizzare la struttura con il comando DESCR:

```
DESCR studenti;
```

- Questo comando mostra l'elenco delle colonne di una tabella, con il loro tipo.
- Il comando DESCR è molto utile per analizzare le tabelle di sistema di oracle Oracle, è verrà utilizzato molto nella parte spaziale del corso.

Risultato In Oracle XE

Descrizione della tabella studenti in Oracle XE



Distruzione di una tabella

- Per distruggere definitivamente una tabella, si utilizza il comando DROP, provate adesso a cancellare la tabella studenti (la rifaremo meglio dopo):

```
DROP TABLE studenti;
```

- Il comando distrugge per sempre la tabella (attenzione ad usarlo con cognizione di causa).
- Abbiamo usato il verbo italiano "distruggere" e non "cancellare" per non confondere le due operazioni: diremo cancellare (DELETE) nel caso in cui vogliamo cancellare i dati di una tabella senza distruggerne la struttura. Il comando DROP invece cancella i dati di una tabella e ne distrugge anche la struttura.
- Attenzione! Non c'e' modo di recuperare una tabella distrutta, a meno che non si sia iniziata una TRANSAZIONE (di cui parleremo più avanti).

Commenti nel codice

- Come in tutti i linguaggi per computer, è possibile inserire in SQL, delle parti di commento che vengono ignorate:
  - Commenti a fine riga: tutto quello che segue il simbolo - - viene ignorato.
  - Commenti multi riga (stile C++): tutto quello che è compreso fra i simboli /* e */.

```
CREATE TABLE studenti
(
  nome VARCHAR2(128),  -- Nome dipendente
  /* Altri dati dello
     Studente */
  eta NUMBER(10),
…
);
```

Specifiche avanzate di una tabella

- Le colonne di una tabella possono avere molte specifiche aggiuntive:
- Ne vediamo due:
  - Specifica della chiave primaria
  - Specifica di campo obbligatorio
- Il nuovo comando di creazione è:

```
CREATE TABLE studenti
(
  nome VARCHAR2 (128) PRIMARY KEY,
  eta NUMBER(3),
  codice_corso NUMBER(6) NOT NULL
);
```

- PRIMARY KEY specifica che il campo nome è quello che identifica univocamente le righe della tabella.
- NOT NULL specifica che il campo codice_corso è obbligatorio e non può rimanere vuoto (NULL) durante l'inserimento delle righe della tabella.
- Provate adesso ha ricontrollare la struttura con il comando DESCR.

Modifiche alla struttura di una Tabella

- Una volta che una tabella è stata creata, sono possibili delle modifiche dinamiche alla sua struttura, tramite il comando ALTER TABLE, ad esempio aggiungiamo la colonna professione:

```
ALTER TABLE studenti
ADD professione VARCHAR2(128);
```

- Provate ad rieseguire DESCR.
- Allo stesso modo possiamo cancellare una colonna con il comando:

```
ALTER TABLE studenti
DROP COLUMN professione;
```

Manipolazione dei Dati

- Vediamo adesso i comandi che permettono di manipolare i dati di una tabella.
- I comandi sono:
  - INSERT : inserisce righe in una tabella
  - DELETE: cancella righe di una tabella
  - UPDATE: modifica i dati di una riga di tabella

Inserimento di dati

- La struttura del comando INSERT è la seguente:

```
INSERT INTO nome_tabella
(nome_colonna1, nome_colonna2, …)
  VALUES (valore1, valore2, …);
```

- Bisogna quindi specificare la tabella, l'elenco delle colonne che vogliamo inserire, quindi l'elenco corrispondente dei valori.

Inserimento Valori studenti

- Proviamo adesso ad inserire alcune righe nella tabella studenti:

```
INSERT INTO studenti
(nome,eta,codice_corso)
VALUES ('Claudio Rocchini', 39, 1);
```

- Notate che il nome è una parola, quindi va fra apici, mentre l'età e il codice del corso (1) sono numeri, quindi senza apicetti.
- Nome, eta e codice_corso sono nomi di colonne, non parole, quindi non vanno fra apici.
- Notate anche le virgole, che separano colonne e valori: ovviamente dopo l'ultimo valore la virgola non ci vuole.
- Niente panico: la sintassi à una brutta bestia, che si doma con l'esperienza.

Inserimento altri valori

- Per rendere più interessante la nostra tabella inseriamo altre righe:

```
INSERT INTO studenti
(nome,eta,codice_corso)
VALUES ('Altair Pirro', 18, 2);
```

- E poi (in una query differente):

```
INSERT INTO studenti
(nome, codice_corso)
VALUES ('Gianfranco Amadio', 1);
```

- Notate che nel caso di Amadio non abbiamo inserito l'età, che non è obbligatoria. Inserite anche i vostri nomi e variate il codice del corso da 1 a 2.

Il valore NULL

- Il campo età non è obbligatorio.
- Quando un dato di una riga non è inserito, la relativa casella nella tabella è vuota.
- Il valore vuoto ha in Oracle un nome: NULL
- Ad esempio potevamo scrivere il comando:

```
INSERT INTO studenti
(nome, eta, codice_corso)
VALUES ('Gianfranco Amadio', NULL, 1);
```

- Intendendo che il campo età deve rimanere vuoto.
- La parola NULL sarà particolarmente utile nei controlli, che vedremo in seguito.

Test sui vicoli (1)

- Nella tabella che abbiamo inserito ci sono due vincoli: la chiave primaria e l'obbligatorietà del campo codice_corso.
- Se proviamo ad inserire una nuova riga con un nome già esistente, es:

```
INSERT INTO studenti
(nome,eta,codice_corso)
VALUES ('Claudio Rocchini', 16, 32);
```

- Otteniamo un errore del *tipo:violata restrizione di unicità.* La chiave primaria infatti deve essere unica.

Test sui vincoli (2)

- Abbiamo visto che il campo età non è obbligatorio.
- E' obbligatorio invece il campo codice_corso.
- Proviamo ad eseguire il seguente comando, per inserire uno studente di cui non conosciamo il corso:

```
INSERT INTO studenti
(nome,eta )
VALUES ('Sconosciuto', 20);
```

- *Otterremo un errore del tipo: impossibile inserire NULL in STUDENTI.CODICE_CORSO*
- Inserire dei controlli nelle tabelle è molto importante per controllare la correttezza e la completezza dei dati.

Cancellazione di Righe

- Il comando DELETE permette di cancellare righe da una tabella.
- La sua forma più semplice sarebbe:

```
DELETE FROM studenti;
```

- **NON ESEGUITE QUESTO COMANDO!**
- Il comando sopra citato cancella TUTTE le righe di una tabella, senza possibilità di recupero.
- La forma che invece spesso si utilizza è:

```
DELETE FROM studenti
WHERE condizione;
```

Cancellazione con condizione

- La specifica di una condizione permette di eliminare solo quelle righe che rispettano la condizione specificata, es:

```
DELETE FROM studenti
WHERE eta=39
```

- Il comando cancellerà tutte le righe che hanno l'età uguale a 39
- Nella condizione è possibile controllare le colonne, eseguire confronti di uguaglianza(=), diversità (<>), confronti di quantità (< e >), ed usare i connettivi logici AND, OR, NOT.

Esempi di condizioni

- Una descrizione delle condizioni logiche esula dagli scopi di questo corso, proponiamo alcuni esempi:

```
… WHERE eta=39 AND codice_corso=1
```

- Cancella gli studenti di 39 anni E che hanno il codice corso uguale a 1

```
… WHERE nome LIKE 'Cl%'
```

- Cancella gli studenti il cui nome inizia per Cl.

Modifica dei dati

- I dati si modificano con il comando UPDATE.
- La struttura del comando UPDATE è:

```
UPDATE nome_tabella
SET nome_colonna = valore
WHERE (condizione)
```

- Dove la struttura della condizione è del tutto uguale al comando DELETE.

Esempio di modifica

- Proviamo adesso a cambiare il codice corso di qualcuno:

```
UPDATE studenti
SET codice_corso = 2
WHERE nome='Claudio Rocchini';
```

- Il valore della colonna specificata viene cambiato per tutte le righe che rispettano la condizione impostata.
- Se non si specifica la condizione, update modifica TUTTE le righe della tabella, settando un un valore costante sulla colonna.

Esempio di modifica 2

- Il valore NULL può essere utilizzato nei controlli, come ad esempio:

```
UPDATE studenti
SET eta=99
WHERE eta= NULL;
```

- Tutte le caselle età vuote (=NULL) vengono riempite con 99.

Un'altra tabella

- Prima di passare all'interrogazione dei dati, creiamo una tabella corsi, che ci servirà per fare degli esempi:

```
CREATE TABLE corsi
(
  codice_corso NUMBER(6) PRIMARY KEY,
  descrizione VARCHAR2(128) NOT NULL
);
```

- Ormai siamo esperti: il codice è la chiave primaria, segue una descrizione testuale obbligatoria.

Altri dati

- Popoliamo la tabella corsi:

```
INSERT INTO corsi
(codice_corso,descrizione)
VALUES (1,'Basi di Dati');
```

- Ed infine un altro corso:

```
INSERT INTO corsi
(codice_corso,descrizione)
VALUES (2,'Sistemi Informativi Territ.');
```

- Possiamo inserire altri corsi, l'importante è che siano presenti tutti i codici di corso che abbiamo inserito per gli studenti.

Altri vincoli: introduzione alle relazioni

- Un database non è fatto solo di entità (tabelle) ma anche di relazioni.
- Due oggetti sono in relazione se esiste un dato che li mette in collegamento.
- Gli studenti sono in relazione con i corsi, dato che per ogni studente abbiamo specificato un codice di corso.
- Le relazioni possono anche essere specificate esplicitamente con l'aggiunta di un vincolo (constraint) alla tabella.

Creare Relazioni

- Digitiamo il seguente comando:

```
ALTER TABLE studenti
ADD CONSTRAINT studenti_corso_fk
FOREIGN KEY (codice_corso) REFERENCES
  corsi(codice_corso);
```

- Il comando esplicita la relazione fra studenti e corsi, ed è formato da un vincolo sulla tabella studenti.
- Studenti_corso_fk è il nome del vincolo.
- Il vincolo afferma che la chiave straniera codice_corso di studenti DEVE riferire una valore della colonna codice_corso, nella tabella corsi.
- Se nella tabella corsi non ci sono tutti i codici necessari, la creazione della relazione è impossibile.

Controllo Relazioni

- Proviamo adesso ad inserire i seguenti dati:

```
INSERT INTO studenti
(nome,codice_corso)
VALUES ('Fantomas',99);
```

- Otteniamo un errore del tipo:*restrizione di integrità violata (STUDENTI_CORSO_FK) : chiave madre non trovata*.
- La congruenza delle relazioni viene controllata dinamicamente in ogni momento: il corso numero 99 non esiste.

Indici

- Supponiamo di prevedere molte ricerche sull'età degli studenti; inoltre supponiamo che gli studenti siano tanti.
- In questo caso sarà opportuno creare un INDICE.
- Gli indici servono per velocizzare le ricerche di valori su una (o più) colonne di una tabella.
- Per creare un indice sulla colonna età della tabella studenti, eseguiamo il comando:

```
CREATE INDEX studenti_eta_idx ON studenti(eta);
```

- Studenti_eta_idx è il nome dell'indice.
- Apparentemente la presenza di un indice non cambia il funzionamento del database. In realtà vedremo che nel caso di dati spaziali, l'indice è fondamentale per la ricerca veloce dei dati.
- Gli indici non vengono mai creati automaticamente: devono essere progettati con cura da chi crea la struttura del db, in funzione del tipo di ricerche da effettuare e dal tipo (e dalla quantità) dei dati presenti.
- Nota: non è mai necessario creare indici per le colonne chiave primaria: in questo caso un indice è creato automaticamente.

Interrogazioni: Introduzione

- Siamo arrivati finalmente alla parte finale di SQL: l'interrogazioni dei dati.
- Sebbene le interrogazioni siano eseguito dall'unico comando SELECT, questo è il comando più complesso.
- Le forme del comando SELECT sono moltissime, quindi ne vedremo alcuni brevissimi esempi.

Forma semplice di SELECT

- La forma più semplice di SELECT è

```
SELECT colonna1,colonna2
FROM tabelle
WHERE (condizioni)
ORDER BY colonna1,colonna2;
```

- Quindi bisogna specificare: le colonne da visualizzare, la tabella sorgente, eventuali condizioni, uguali a quelle dei comandi UPDATE e DELETE, ed un eventuale ordine (l'ordine non è mai definito per default).
- Invece di scrivere un elenco di colonne è possibile scrivere il simbolo * che indica tutte le colonne della tabella.

SELECT: Tabelle intere

- Per visualizzare un'intera tabella, scrivete il comando:

```
SELECT *
FROM studenti;
```

- Oppure:

```
SELECT *
FROM corsi
ORDER BY descrizione;
```

- Nel secondo caso i corsi saranno ordinati per descrizione (ordine alfabetico).

SELECT: Alcune colonne

- Per visualizzare un sottoinsieme di colonne, possiamo scrivere:

```
SELECT nome,eta
FROM studenti
ORDER BY eta;
```

- In questo caso l'ordine è per età crescente.

SELECT: filtro

- Possiamo scegliere di visualizzare solo alcune righe di una tabella:

```
SELECT nome
FROM studenti
WHERE eta < 60
AND    codice_corso=1;
```

- Che in italiano si legge: seleziona il nome degli studenti che hanno un età minore di 60 anni e appartengono al corso 1.

Aggregazioni: Schema

- Un secondo tipo di SELECT, è quella del tipo "aggregante", in cui più linee di una tabella possono essere aggregate insieme, da particolari funzioni, lo schema è:

```
SELECT funzioni_aggreganti
FROM tabella
WHERE (condizione sulle righe)
GROUP BY colonne
HAVING (condizione sul risultato aggreg.)
```

- Le condizioni sono sempre nella stessa forma.

Funzioni di Aggregazione

- Esempi di funzioni di Aggregazione sono:
  - Min (minimo)
  - Max (massimo)
  - Avg (media)
  - Sum (somma)
  - Count (numero di…)
  - …
- Vedremo che ci sono anche funzioni di aggregazione spaziale (es. baricentro di insieme di oggetti).

Esempio di aggregazione (1)

- Scriviamo:

```
SELECT min(eta),max(eta),avg(eta)
FROM studenti;
```

- Il risultato saranno i valori minimo, massimo e medio di tutte le età della tabella studenti.
- Si noti che il risultato in questo caso è una sola riga: tutte le righe della tabella studenti sono state aggregate in una.

Esempio di aggregazione (2)

- Scriviamo adesso:

```
SELECT codice_corso, min(eta),max(eta),avg(eta),count(*)
FROM studenti
GROUP BY codice_corso;
```

- La query è simile alla precedente, ma in questo caso le righe non sono aggregate tutte insieme, ma raggruppate secondo il codice del corso.
- Il risultato è l'analisi dell'età degli studenti a secondo del corso a cui appartengono.
- Il risultato in questo caso è formato da più righe: una per ogni corso presente.
- La scritta count(*) conta il numero di righe corrispondenti, vale a dire il numero di partecipanti ad un corso. Le colonne specificate nella funzione count sono ignorate, dato che questa funzione conta le righe.

Join: introduzione

- Nel nostro database abbiamo due tabelle: studenti e corsi. Nella tabella studenti è presente il nome dello stesso e il codice del corso. Nella tabella corsi è presente la descrizione.
- Vogliamo adesso visualizzare il nome di ogni studente con associata la descrizione del corso seguito.
- Per fare questo è necessario utilizzare la relazione che intercorre fra le due tabelle: il termine tecnico di questa operazione è JOIN.

Esecuzione di una join

- Colleghiamo le due tabelle con la query:

```
SELECT studenti.nome,
       corsi.descrizione
FROM  studenti,corsi
WHERE studenti.codice_corso =
      corsi.codice_corso;
```

- Ci sono alcuni particolari da notare:
- Per prima cosa in questa query facciamo utilizzo di DUE tabelle: dopo FROM infatti possiamo utilizzare quante tabelle vogliamo.
- In secondo luogo vediamo che le colonne dopo la select sono specificate nella forma NOME_TABELLA.NOME_COLONNA: questa specifica è necessaria in presenza di più tabelle per chiarire da quale tabella si pesca la colonna. Codice_corso ad esempio è presente in entrambe le tabelle ed Oracle non sa decidere di quale tabella fa parte.

Join: vincolo

- Infine analizziamo la clausola where: è questa che mette in relazione concretamente le due tabelle.
- Senza la clausola WHERE (provate a cancellarla ed eseguire la query), Oracle esegue quello che si chiama “prodotto cartesiano” dei valori, vale a dire tutte le combinazioni possibili fra studenti e corsi.
- La clausola where invece, fra tutte le combinazioni, seleziona solo quelle in relazione.
- Le join fra tabelle sono molto importanti nel campo spaziale: vedremo che lo stesso meccanismo può essere utilizzato per creare relazioni spaziali (es. relazionare gli edifici con le strade a seconda della minima distanza relativa).

Viste

- Una volta che abbiamo creato una SELECT interessante, è possibile che ci serva più volte. E’ possibile salvare le query come viste, provate:

```
CREATE VIEW stud_corsi AS
SELECT studenti.nome,
               corsi.descrizione
FROM studenti,corsi
WHERE studenti.codice_corso = corsi.codice_corso;
```

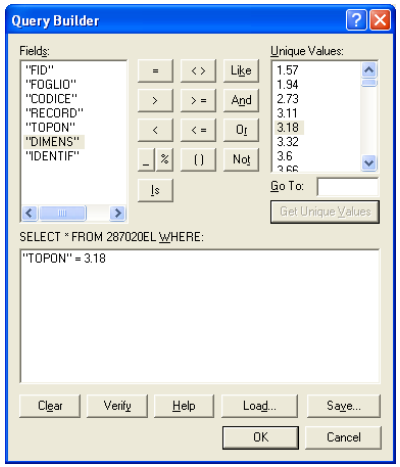
- Le viste sono query con nome. Una volta create si utilizzano come se fossero tabelle. In realtà i dati delle viste variano al variare delle tabelle sottostanti (studenti e corsi).
- Provate ad eseguire query su stud_corsi

Per concludere SELECT

- Questa è solo una brevissima introduzione A SQL.
- La struttura del comando SELECT ha molte altre possibilità, che richiederebbero almeno un anno di corso.
- Il linguaggio SQL è di per sé molto semplice, ma la creazione di un comando SELECT non banale, richiede, in alcuni casi, una certa esperienza.

Un inciso: Sistemi grafici di costruzione query:

- SQL è un linguaggio molto elegante, ed è chiaro di per sé.
- Molti sistemi prevedono però un ausilio grafico alla costruzione della query.
- Vediamo ad esempio il Query Builder di ArcGis.
- Alla fine si ottiene sempre la query SQL.



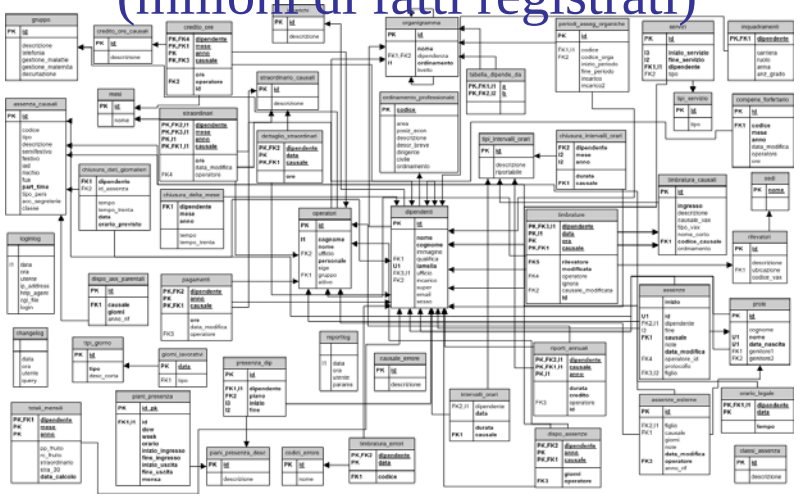
Nota sulla configurazione del Corso

- Per scopi dididattici, ciascuno di voi ha installato il proprio server personale.
- In realtà l'utilizzo di Oracle come database interprise sarebbe stato quello di installare un server unico, con i dati condivisi, su cui tutti lavoravano in modo concorrente.
- Si è deciso un'installazione separata per ciascuno per evitare di dover fare una serie di utenti diversi.
- I nomi degli indici poi sono globali per ogni database, quindi serebbero andati in conflitto.
- Oracle è studiato per un utilizzo contemporaneo dei dati da parte di molti utenti (anche in modifica).

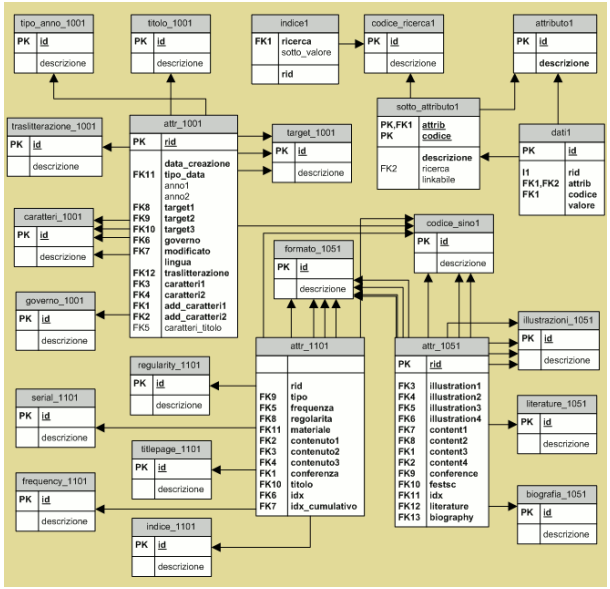
I DB reali

- I database reali (ovviamente) possono essere molto complessi.
- Vediamo una panoramica di alcuni semplici schemi di db creati dall'autore.

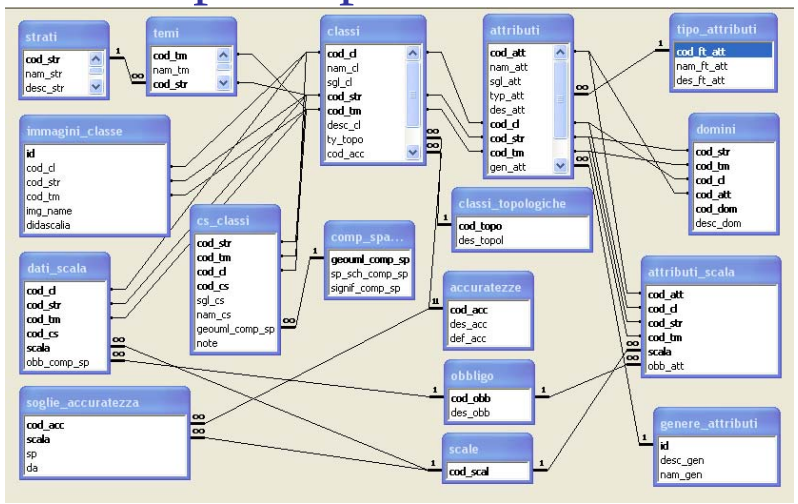
Esempi: Gestione Amm. IGM (milioni di fatti registrati)



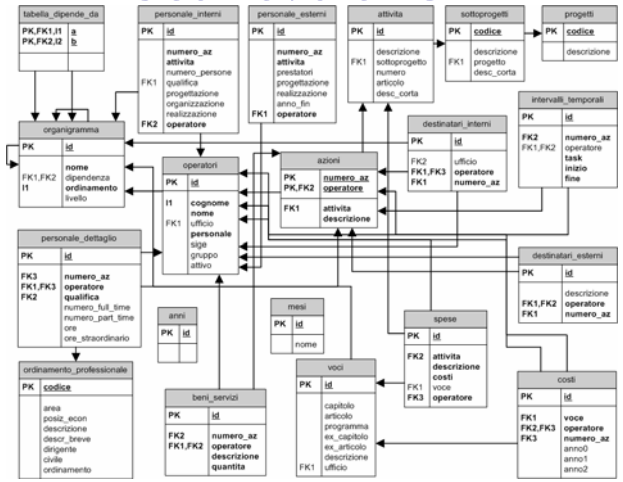
Esempio: Biblioteca IGM (100000 volumi)



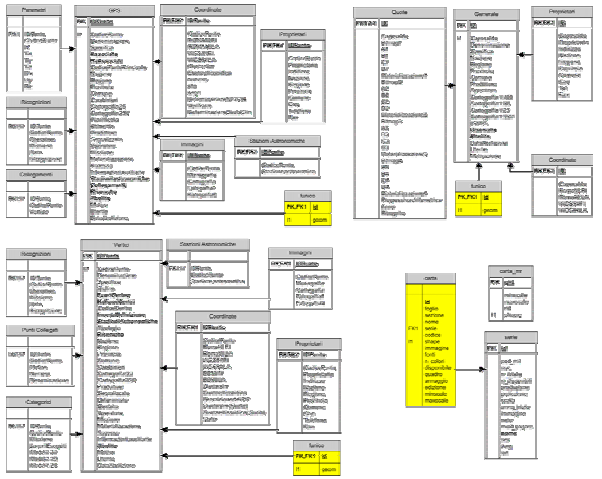
Esempio: Specifiche Intesa



Esempio: Piano Pluriennale Scorrevole IGM



Esempio: Punti Trigonometrici



Esempi di query amministrativa

```
SELECT pd.dipendente,
       ol.tempo - (CASE WHEN g.tipo=0 THEN pp.orario ELSE INTERVAL '00:00' END) as tempo,
                   CASE WHEN ol.tempo IS NULL THEN NULL
                        WHEN ol.tempo-(CASE WHEN g.tipo=0 THEN pp.orario ELSE INTERVAL '00:00' END) >=
                             INTERVAL '00:30'
                        THEN ol.tempo-(CASE WHEN g.tipo=0 THEN pp.orario ELSEINTERVAL '00:00' END)
                        ELSE INTERVAL '00:00'
                    END as tempo_trenta,
                    g.data,
                   (SELECT a.causale
                    FROM assenze AS a, assenza_causali AS ac
                    WHERE a.dipendente = pd.dipendente
                                         AND   a.causale=ac.id
                    AND   a.inizio <= g.data
                    AND   (a.fine IS NULL OR a.fine >= g.data)
                                         AND  ( g.tipo=0 OR (g.tipo=1 AND ac.festivo) OR (g.tipo=2 AND
                                              ac.semifestivo) )
                  ) AS assenza,
                  pp.orario
              FROM giorni_lavorativi as g LEFT OUTER JOIN orario_legale AS ol ON (ol.data = g.data AND ol.dipendente =
                   [param:d_id]),
                   presenza_dip as pd,
                   piani_presenza as pp
              WHERE date_part('year'::text ,g.data)=[param:anno]
              AND   pd.dipendente=[param:d_id]
              AND   pd.inizio<=g.data
              AND   pd.fine>=g.data
              AND   (pp.dow  = -1 OR pp.dow::double precision = date_part('dow', g.data))
              AND   (pp.week = -1 OR pp.week = (date_part('week', g.data)::integer % 2))
              AND   pd.piano = pp.id;
```