Data Base

Note

Data (types) → structured
Data (types) → Unstructured

Data → Storage
Data → Retrieval

- DBMS = Data Base Management System ⇝ Oracle
  - Connection of interrelated data
  - Set of programs to access data
  - convenient and efficient to use
- The need for databases → Accessing various datas → Banking
  - → Airlines
  - → Sales
  - → Universities
  - → human resources
  - → Manufacturing
  - → Online shoping

- First, databases were file systems.

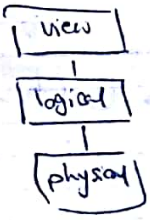
→ problems:

1- Data Redundancy & incosistency
2- Difficulty in accessing Data
3- Data isolation
4- Integrity problem
5- Atomicity of updates
6- Concurrent Access by multiple users
7- Security problems

- Database = collection of interrelated data and a set of programs that allows users to access and modify these data

↳ purposes → providing an abstract view of the data
↳ purposes → hiding details of how data are stored and maintained

- Levels of Abstraction
  - physical : how data is stored actually
  - logical : what data, relationship between data
  - view : hiding certain info from certain users



Data Model = structure of dbs describing
- Data
- " relationships
- " semantics
- " constraints

- → Relational → tables
- → Entity-Relation ✓✓✓
- → Object-based
- → Semistructured (XML)

ملاحظات

فروردین

ش	ی	د	س	ج	ب	ج
			۱	۲	۳	۴
۵	۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸
۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵
۲۶	۲۷	۲۸	۲۹	۳۰	۳۱	

②
- Schema = Overall design of db → physical / logical (≈ type of variable)
- Instance = the actual content of db (≈ value of variable)
- physical data independence
- DDL = notation for db schema
  - Domain Constraints
    جنس دیتای کل ستون
  - Referential Integrity
    ارجاع به جداول دیگر
  - Assertions
    اعمال محدودیت روی ستون‌ها
  - Authorization
    چه کسی چه نوع دسترسی روی چه جدولی داره

Data of Data
↑
- Data Dictionary contains metadata that describes info about db
- DML ≈ query language $\xrightarrow{\text{types}}$
  - Procedural = what data & how to get it
  - Declarative = what data (No matter how to get it)

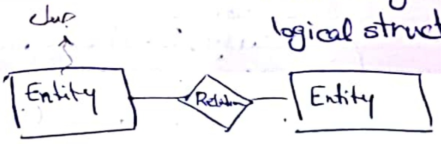
types of access:
- insert info
- delete info
- Modify info
- Retrieve info

Language classes:
- pure $\xrightarrow{\text{e.g.}}$ Relational Algebra
- commercial $\xrightarrow{\text{e.g.}}$ SQL

- Database design
  - logical : Schema
    - Business decesion : what attributes
    - CS decesion : what relation
  - Physical
- Design Approaches (for "good" relation design) : 1- Entity Relation Model
  2 - Normalization Theory
- Specification of Functional Requirements : kinds of operations that can be performed on data
  ↳ (delete? insert? search? ...)

①
- Entity - Relationship Model
  ↳ Relation between objects (entities)
  ↳ described by a set of attributes

اول اصلا به روابط کار نداریم، موجودیت ها رو مشخص می کنیم (استاد، دانشجو، دانشکده، ...)
بعد میایم به روابط بین این موجودیت ها می پردازیم



- UML = Unified Modeling Language
  ↓
  used for graphially manifesting the logical structures

ملاحظات

فروردین

ش	ی	د	س	چ	پ	ج
			۱	۲	۳	۴
۵	۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸
۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵
۲۶	۲۷	۲۸	۲۹	۳۰	۳۱	

پنجشنبه
۱۰
فروردین
30 MARCH 2017
۱ رجب ۱۴۳۸

Note

- ② Normalization

اولش همینجوری تمام داده ها رو میریزی رو میز،
بعد مرتب می کنی با توجه به دنیای واقعی و
مثال نقض هایی که پیدا می کنی، جدول رو می شکونی و
سعی می کنی به یه شمای نرمال برسی.
که بازیابی داده و ذخیره اطلاعات توش راحته.
(فاقد افزونگی بیهوده)

problems of a bad design { Repetition of info
Inability to represent certain info

- Storage Manager : interface between low-level data stored in db & (application & queries)

★ tasks : { Interaction with OS file manager
Efficient { store / retrieve / update } of data

★ Includes
- → Transaction manager = handling failure or concurrent transactions
- → File manager = handling allocation of space / data structures
- → Buffer Manager = fetch data into main memory / handle large-sized data
- → Authorization / Integrity manager = authority of users / satisfaction of constraints
  (Integrity: درستی داشتن)

★ Data Structures in physical system implementation → { Data files = stores db itself
Data Dictionary = stores metadata
Indices = fast access to data items.

ولادت حضرت امام محمد باقر (ع) (۵۷ هـ ق)

جمعه
۱۱
فروردین
31 MARCH 2017
۲ رجب ۱۴۳۸

- Query Processing { 1. Parsing and translation (≈ کامپایل)
2. Optimization
3. Evaluation (≈ اجرا)

- Transaction manager = keeps db in a consistent state, despite { system failure / transaction failure

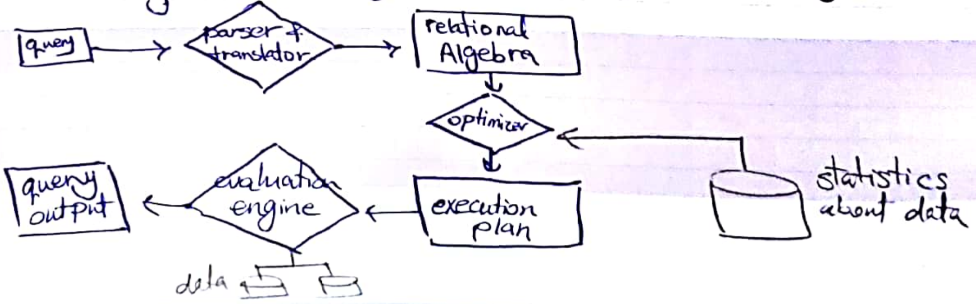
★ Recovery manager = restore the state of db before the failure

★ Concurrency-control manager = consistency of db during concurren transactions

فروردین

ش	ی	د	س	چ	پ	ج
			۱	۲	۳	۴
۵	۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸
۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵
۲۶	۲۷	۲۸	۲۹	۳۰	۳۱	

ملاحظات



- Data Mining : semi-auto analyzing db to find useful patterns.
- Info retrieval : querying of unstructured textual data
- Database Architecture is influenced by underlying computer systems.
  - → Centralized
  - → Client-Server
  - → Parallel
  - → Distributed

- History of db systems

★ 1950 , 1960 : magnetic tapes , punched cards

★ 1960 , 1970 :
- Hard disk (direct access to data)
- Widespread use of Network
- high-performance transaction processing
- relational data model

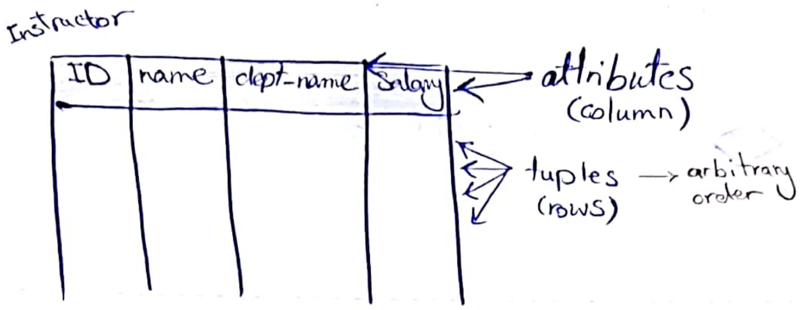
★ 1980 :
- Object-oriented database System
- Parallel and distributed db "
- Relational evolves to commercial (SQL)

★ 1990 :
- Web commerce
- data-mining apps
- multi-terabyte data warehouses

Early
★ 2000 : XML and XQuery

③

Note



- Domain = allowed values for att.s
- Att.s should be atomic $\xrightarrow{e.g.}$ "9631046" do not use for entrance year
- null ∈ every domain
  ↳ ≡ unknown value ⇝ causes complication

	definition	example
K ← superkey	sufficient to identify a unique tuple of a relation	{ID}, {ID, name}, ...
candidate key	if K is minimal	{ID}
primary key	One of the candidate keys	
foreign key	When value in one relation appears in another (Must be primary in it's own table)	dept-name

- Relational Algebra

مثال →

☆ Selection of Rows	σ	/ Select
☆ Selection of columns	Π	/ Project
☆ Union of relations	$\cup$	
☆ Set difference of relations	$-$	
☆ Set ✓ Intersection of relations	$\cap$	
☆ Joining two relations (Cartesian Product)	$\times$	
☆ Rename	$\rho_x(E)$	→ returns the expression E under the name X
☆ Natural Join	$\bowtie$	

روز طبیعت(تعطیل)

ملاحظات

* Each query input is a table.
Also, " " output " " ".

فروردین

ش	ی	د	س	چ	پ	ج
			۱	۲	۳	۴
۵	۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸
۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵
۲۶	۲۷	۲۸	۲۹	۳۰	۳۱	

<> → not equal to

Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	σ salary > = 85000 (instructor)
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	Π *ID, salary* (instructor)
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	Π *name* (instructor) ∪ Π *name* (student)
	Output the union of tuples from the *two* input relations.
− (Set Difference)	Π *name* (instructor) − Π *name* (student)
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

ش	ی	د	س	چ	پ	ج
			۱	۲	۳	۴
۵	۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸
۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵
۲۶	۲۷	۲۸	۲۹	۳۰	۳۱	

```
4
>> Types:
    char(n).  Fixed length character
    varchar(n).  Variable length with maximum length n.
    int.  Integer
    smallint.  Small integer
    numeric(p,d).  Fixed point number, with user-specified precision of p digits,
with d digits to the right of decimal point.
    real, double precision.  Floating point and double-precision floating point
numbers
    float(n).  Floating point number at least n digits.

>>
    create table instructor (
        ID          char(5),
        name        varchar(20) not null,
        dept_name   varchar(20),
        salary      numeric(8,2),
        primary key (ID, name),
        foreign key (dept_name) references department,
        foreign key (salary) references teacher
    );

# primary key automatically not null

>> Remove all rows from the student table
    delete from student

>> Remove table
    drop table r

>> Add attribute A with Domain D to table r
    alter table r add A D

>> Delete attribute A fro table r
    alter table r drop A

>> Remove duplicated values of A in r
    select distinct A
    from r

>> Not remove duplicated values of A in r
    select all A
    from r

>> select all attributes of r
    select *
    from r

>> Create a table with attribute A and has a row that has 437 as a value
```

```
    select "437" as A

>> Result is a table with one column and N rows (number of tuples in the
instructors table), each row with value “A”
    select  "A"
    from r

>> would return a relation that is the same as the instructor relation, except that
the value of the attribute salary is divided by 12
    select salary / 12
    from instructor

>> Rename A to B
    select A as B = select A B

    from instructor as T, instructor as S

# WHERE is for adding condition to the queries that can be combined using the
logical connectives and, or, not
# The from clause lists the relations involved in the query

>> Cartesian product of A and B
    from A, B

>> These queries are equal (just product the rows that has the same value of the
same keys)
    select *
    from instructor, teaches
    where instructor.ID = teaches.ID
    =
    select *
    from instructor natural join teaches

>> like uses patterns that are described using two special characters (% and _)
(Patterns are case sensitive):
    percent ( % ). matches any substring.
    underscore ( _ ). matches any character.
    note:
        Match the string “100%”: like ‘100\%'  escape  '\'

    select name
    from instructor
    where name like '%dar%'

>> Ordering the Display of Tuples (desc for descending order or asc for ascending)
    select A
    from r
    order by A desc

    Can sort on multiple attributes:
```

```
        order by A, B

>> A < x < B
    where x between A and B

>> Tuple comparison
    (A.ID, text) = (B.ID, "hi")

>> duplicates :?

>> Set Operations
    or
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        union
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    and
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        intersect
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    not
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        except
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    # To retain all duplicates use the corresponding multiset versions union all,
intersect all and except all.

# Any comparison with null returns unknown that it is treated as false

>> Aggregate Functions
    avg: average value
    min: minimum value
    max: maximum value
    sum: sum of values
    count: number of values
```

```
        select count (distinct ID)
        from teaches
        where semester = ’Spring’ and year = 2010;

    >> Attributes in select clause outside of aggregate functions must appear in
group by list
        select dept_name, avg (salary)
        from instructor
        group by dept_name;

    >> Having Clause: predicates in the having clause are applied after the
formation of groups whereas predicates in the where clause are applied before
forming groups
        select dept_name, avg (salary)
        from instructor
        group by dept_name
        having avg (salary) > 42000;

    # All aggregate operations except count(*) ignore tuples with null values on
the aggregated attributes

>> Nested Subqueries: A subquery is a select-from-where expression that is nested
within another query.
    >> in the Where Clause
        >> Set Membership: in
            select distinct course_id
            from section
            where semester = ’Fall’ and year= 2009 and
                course_id in (select course_id
                            from section
                            where semester = ’Spring’ and year= 2010);

        >> Set Comparison: some
            select name
            from instructor
            where salary > some (select salary

                                from instructor

                                where dept name = ’Biology’);

        >> Set Comparison: all
            select name
            from instructor
            where salary > all (select salary
                                from instructor
                                where dept name = ’Biology’);

        >> Test for Empty Relations: exists returns true if subquery is nonempty
            select course_id
```

```
            from section as S
            where semester = ’Fall’ and year = 2009 and
                exists (select *
                        from section as T
                        where semester = ’Spring’ and year= 2010
                            and S.course_id = T.course_id);

        >> Test for Absence of Duplicate Tuples: unique returns true if subquery
contains no duplicates
            select T.course_id
            from course as T
            where unique (select R.course_id
                        from section as R
                        where T.course_id= R.course_id

                            and R.year = 2009);

    >> in the From Clause
        select dept_name, avg_salary
        from (select dept_name, avg (salary)
                from instructor
                group by dept_name) as dept_avg (dept_name, avg_salary)
        where avg_salary > 42000;

        # do not need to use the having clause

    >> in the Select Clause: Subqueries in select clause most return a scalar
        select dept_name,
            (select count(*)
            from instructor
            where department.dept_name = instructor.dept_name) as num_instructors
                from department;

>> with clause: provides a way of defining a temporary relation
    with max_budget (value) as
        (select max(budget)
        from department)
    select department.dep_name
    from department, max_budget
    where department.budget = max_budget.value;

>> Deletion
    >> delete table
        delete from r

    >>
        delete from instructor
        where dept_name= ’Finance’;

>> Insertion
```

```
    >>
        insert into course
            values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

    >>
        insert into course (course_id, title, dept_name, credits)
            values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

    >>
        insert into student
            values (’3003’, ’Green’, ’Finance’, null);

    >>
        insert into student
            select ID, name, dept_name, 0
            from instructor

>> Updates
    >>
        update instructor
        set salary = salary * 1.03
        where salary > 100000;
```

```
5
>> Join Operations:
        >> Natural Join = Natural Inner Join:
                A natural join B
                A natural inner join B
                =
                A inner join B on
                A.id = B.id

        >> Outer Join: join then adds tuples from a relation that doesn't match tuples in other relation & uses null values
                >> Left:
                        A natural left outer join B
                        ==
                        A left outer join B on
                        A.id = B.id
                >> Right:
                        A natural right outer join B
                        =
                        A right outer join B on
                        A.id = B.id
                >> Full:
                        A natural full outer join B
                        =
                        A full outer join B on
                        A.id = B.id

        >> Join Conditions:
                natural
                on <predicate> --> condition (like where clause)
                using (Mutual Column)

>> View: provides a mechanism to hide certain data from the view of certain users (virtual relations)\
        create view faculty as
        select ID, name, dept_name
        from instructor

        select name
        from faculty
        where dept_name = ‘Biology’

        # You can create a view from another view

        # You can update views only under these conditions (and the table will be updated too):
                The from clause has only one database relation.
                The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
                Any attribute not listed in the select clause can be set to null
```

The query does not have a group by or having clause.

>> Materialized view: if the actual relations change, the view is kept up-to-date.
by these ways:
View maintenance can be done immediately when any of the relations on which the view is defined is updated.
View maintenance can be performed lazily, when the view is accessed.
Some systems update materialized views only periodically

>> Transaction: a sequence of queries and/or update statements which are Atomic
>> begin implicitly and ended by one of the following:
* Commit work - commits the current transaction then a new will start
* Rollback work - causes the current transaction to be rolled back and undoes updates (restore)

>> Integrity Constraints: guard against accidental damage
>> Not Null
>> Unique: states that attributes form a candidate key
>> Check (a Predicate)
>> Referential Integrity: ensure that a value appears in a relation for a set of attributes also appears for a certain set of attributes in another relation

```
create table section (
        course_id varchar (8),
        sec_id varchar (8),
        semester varchar (6) not null, check (semester in (’Fall’, ’Winter’, ’Spring’, ’Summer’),
        year numeric (4,0),
        building varchar (15),
        room_number varchar (7),
        time slot id varchar (4),
        primary key (course_id, sec_id, semester, year),
        foreign key (course_id) references course,

        check (year > 1759 and year < 2100)
);
```

>> Cascading Actions: delete/update on the referenced relation cascades on the referencing relation
on delete/update: cascade, set null, set default

```
create table course (
        …
        dept_name varchar(20),
        foreign key (dept_name) references department

                on delete cascade
                on update cascade,
        …
)
```

```
        >> Complex Check Clauses:
                # subquery in check clause not supported
                create assertion credits_earned_constraint check
                (not exists (select ID
                                         from student
                                         where tot_cred <> (select sum(credits)

from takes natural join course

where student.ID= takes.ID and

grade is not null and grade<> ’F’)
                                        )
                )
>> Built-in Data Types:
        date ‘2005-7-27’
        time ‘09:00:30’
        timestamp  ‘2005-7-27 09:00:30.75’
        interval:  period of time

>> Default Values:
        create table student
                (ID varchar (5),
                name varchar (20) not null,
                dept_name varchar (20),
                tot_cred numeric (3,0) default 0,
                primary key (ID)
        )

        insert into student(ID,name,dept_name)
                values(’12789’, ’Newman’, ’Comp. Sci.’);

>> Index Creation:
        create table student
                (ID varchar (5),
                name varchar (20) not null,
                dept_name varchar (20),
                tot_cred numeric (3,0) default 0,
                primary key (ID)
        )

        create index studentID_index on student(ID)

>> Large-Object Types: Large objects (photos, videos, CAD files, etc.) are stored
as a large object
        blob: binary large object
        clob: character large object

        B clob(10KB)
        A blob(10MB)
```

```
        # When a query returns a large object, a “locator” is returned that can use to fetch the large object in small pieces, rather than all at once

>> User-Defined Types: two forms --> 1. distinct types 2. structured data types

        create type Dollars as numeric (12,2) final

        create table department
                (dept_name varchar (20),
                building varchar (15),
                budget Dollars
        );

        # Values can be cast to another domain:
                cast (department.budget  to numeric (12,2))

>> Domains:
        # Types and domains are similar. Domains can have constraints

        create domain person_name char(20) not null

        create domain degree_level varchar(10)
                constraint degree_level_test
                        check (value in (’Bachelors’, ’Masters’, ’Doctorate’));

>> Create Table Extensions: Creating tables that have the same schema as an existing table.

        create table temp_instructor like instructor

        create table t1 as
                (select *
                from instructor
                where dept_name= ’Music’)
        with data

>> Authorization
        >> Privilege
                Select(Read) : read access to relations or query in the views
                Insert
                Update
                Delete

        >> Forms of authorization to modify:
                Index - allows creation and deletion of indices.
                Resources - allows creation of new relations.
                Alteration - allows addition or deletion of attributes in a relation.
                Drop - allows deletion of relations
```

```
        >> Grant: to confer authorization
                grant <privilege list>
                on <relation name or view name>
                to <user/role list>

                # <user list> is: 1. a user-id 2. public

                grant update on instructor to U1, U2, U3

                # The authorization may be to only some attributes but not on specific tuples
                        grant update (name) on instructor to U1, U2, U3

        >> Revoke: to revoke authorization
                revoke <privilege list>
                on <relation name or view name>
                from <user/role list>

        >> Roles: Authorizations can be granted to roles as they are granted to users
                create role lecturer;
                grant lecturer to U1;
                grant select on takes to lecturer;

                # U1 inherits all privileges of lecturer
                # We can have Chain of roles

        >> Views:
                # gives us the possibility to define authorization to some specific tuples
                create view geo_instructor as
                        (select *
                        from instructor
                        where dept_name = ’Geology’);

                grant select on geo_instructor to  geo_staff

                Then a geo_staff member can issue: select * from geo_instructor;

        >> References: privilege to create foreign key
                grant reference (dept_name) on department to U1;

        >> Transfer
                >> with Grant option:
                        grant select on department to U1 with grant option;

                >> Cascade: is default
                        revoke select on department from U1, U2 cascade;
```

```
>> Restrict: prevent cascading revocation
        revoke select on department from U1, U2 restrict;
```

6

>> Accessing SQL From a Programming Language

- >> API

- >> calls to:
  - Connect to DB
  - Send SQL commands
  - Fetch results

- >> approaches:
  - Dynamic SQL: in run time
    - JDBC (Java Database Connectivity)
    - ODBC (Open Database Connectivity)
  - Embedded SQL: in compile time

- >> Dynamic SQL:
  - >> JDBC:
    - a java API that support SQL --> for query, update, retrieving and ...

    - by these models:
      - Open a connection
      - Create a “statement” object
      - Execute queries using the Statement object to send queries and fetch results
      - Exception mechanism to handle errors

    - >> Database Connection:
      - JDBC driver that must be dynamically loaded to access the database from Java
        - This is done by invoke: Class.forName("oracle.jdbc.driver.OracleDriver");
          - implementing the java.sql.Driver interface

    - >> Connecting to the Database: open connection
      - getConnection method of DriverManager class
        - Connection conn = DriverManager.getConnection
          - ("Server URL", "User ID", "Password");

    - >> Methods for executing a statement:
      - >> executeQuery: when statement is a query and it has a result
      - >> executeUpdate: when statement is nonquery and it hasn't result --> Update, Insert, Delete, Create Table
        - It returns number of tuples
        - or
        - return zero in DDL statements

>> Retrieving the Results

Retrieving the set of tuples in the result into a ResultSet object

Fetching the results one tuple at a time

Using the next method on the result set to test whether there remains unfetched tuple in the result

>> Attributes are retrieved using various methods names begin with get

getString: retrieve basic SQL data types

getFloat

>> Possible argument get methods

attribute name as a string

An integer indicating the position of attribute within tuple

The statement and connection are both closed at the end of the Java program because there is a limit on number of connections

>> Prepared Statements:

Creating a prepared statement which some values replaced by “?”

actual values will be provided later

Compiling query when it is prepared

```
PreparedStatement pStmt = conn.prepareStatement
                                        ("insert into instructor values(?,?)");
pStmt.setString(1, "88877");
pStmt.setString(2, "Perry");
pStmt.executeUpdate();
pStmt.setString(1, "88878");
pStmt.executeUpdate();
```

>> Metadata Features:

Capturing metadata about: 1. Database 2. ResultSet (relations)

```
ResultSetMetaData rsmd = rs.getMetaData();
for(int i = 1; i <= rsmd.getColumnCount(); i++){
    System.out.println(rsmd.getColumnName(i));
    System.out.println(rsmd.getColumnTypeName(i));
}
```

>> ODBC: Open DataBase Connectivity (ODBC) standard

open a connection with a database, send queries and updates, get back results.

- >> Embedded SQL
  - EXEC SQL statement use to request preprocessor

    ```
    EXEC SQL <embedded SQL statement >;
    ```

    \#

    In some languages, like COBOL, the semicolon is replaced with END-EXEC

    In Java embedding uses

    ```
    # SQL { …. };
    ```

  - >> Database Connection

    ```
    EXEC-SQL connect to server user user-name using password;
    ```

  - >> Variables

    Variables of host language can use in embedded SQL statements.

    preceded by a colon (:) to distinguish from SQL variables (:credit_amount)

    Variables must be declared in DECLARE section

    ```
    EXEC-SQL BEGIN DECLARE SECTION;
            int credit-amount;
    EXEC-SQL END DECLARE SECTION;
    ```

  - >> SQL Query

    ```
    declare c cursor for  <SQL query>
    # c is used to identify the query

    EXEC SQL
            declare c cursor for
            select ID, name
            from student
            where tot_cred > :credit_amount
    END_EXEC
    ```

    - >> open statement: again execute the query:

      ```
      EXEC SQL open c;
      ```

    - >> fetch statement: Placing the values into host language variables

      ```
      EXEC SQL
              fetch c into :si, :sn

      END_EXEC
      ```

    - >> close statement: delete the temporary relation that holds the result of the query

```
                                EXEC SQL close c;

                        >> update:
                                EXEC SQL < any valid update, insert, or delete>;


                                EXEC SQL
                                        declare c cursor for
                                                select *
                                                from instructor
                                                where dept_name = ‘Music’

                                                for update;

                                EXEC SQL
                                        update instructor
                                                set salary = salary + 1000

                                                where current of c;
```

>> Functions and Procedures: Functions/procedures can be written in SQL, or in an external programming language

```
        create function dept_count (dept_name varchar(20))
                returns integer
                begin
                        declare d_count integer;
                                select count (* ) into d_count

                                from instructor
                                where instructor.dept_name = dept_name

                        return d_count;
                end

        select dept_name, budget
        from department
        where dept_count (dept_name) > 12

        # Compound statement: begin … end
        # returns: variable-type that returned
        # return: values that are returned as result

        >> Table Functions: functions that return a relation as a result
                create function instructor_of (dept_name char(20))
                        returns table (
                                ID varchar(5),
                                name varchar(20),
                                dept_name varchar(20),
                                salary numeric(8,2))
                        return table
```

```
                                (select ID, name, dept_name, salary
                                from instructor
                                where instructor.dept_name =
instructor_of.dept_name)

                select *
                from table (instructor_of (‘Music’))
```

>> Triggers: a statement that executed automatically by system as a side effect of modification to database

we must:

1. Specify conditions which the trigger is to be executed.
2. Specify actions to be taken when trigger executes.

events: insert, delete or update

Triggers on update can be restricted to specific attributes

after update of takes on grade

Values of attributes before and after an update can be referenced

referencing old row as   : for deletes and updates

referencing new row as : for inserts and updates

Triggers can activated before an event, which can serve as extra constraints.

For example: convert blank grades to null.

>> Using set Statement

```
        create trigger setnull before update on takes
        referencing new row as nrow
        for each row
        when (nrow.grade=’ ’)
        begin atomic
                set nrow.grade=null;
        end;
```

>> Maintain Referential Integrity

```
        create trigger timeslot_check1 after insert on section
        referencing new row as nrow
        for each row
        when (nrow.time slot_id not in (
                select time slot_id
                from time slot))
        begin
                rollback
        end;

        create trigger timeslot_check2 after delete on timeslot
        referencing old row as orow
        for each row
```

```
when (orow.time slot id not in (
                select time slot_id
                from time slot)
        and orow.time_slot_id in (
                select time_slot_id
                from section))
begin
        rollback
end;
```

\>> Maintain credits_earned value

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> ’F’ and nrow.grade is not null
        and (orow.grade=’F’ or orow.grade is null)
begin atomic
        update student
        set tot cred=tot cred+
                (select credits
                from course
                where course.course_id=nrow.course_id)
        where student.id = nrow.id;
end;
```

\>> Triggers can be disabled by (default is enable):

```
alter trigger trigger_name disable
disable trigger trigger_name
```

\>> A trigger can be dropped

```
drop trigger trigger_name
```

\>> Triggers were used for: 1. maintain summary, 2. support for replication and now DBs havr built-in support for these

Encapsulation facilities can be used instead of triggers in cases :

1.Define methods to update fields
2. Carry out actions as part of the update methods instead of through a trigger

\>> Risk of unintended execution of triggers

for example, when

- Loading data from a backup copy
- Replicating updates at a remote site
- Error leading to failure of critical transactions that set off the trigger
- Cascading execution

Trigger execution can be disabled before such actions.

```
        >> Ranking
                select ID, rank() over (order by GPA desc) as s_rank
                from student_grades
                order by s_rank

                >> Ranking with Partitions
                        select ID, dept_name, rank () over (partition by dept_name
order by GPA desc)
                                                                as dept_rank
                        from dept_grades
                        order by dept_name, dept_rank;

                # Ranking is done after applying group by
                # limit n: Top n Items
                # ntile: takes the tuples in each partition and divides them into n
buckets
                        select ID, ntile(4) over (order by GPA desc) as quartile
                        from student_grades;

                # nulls first or nulls last
                        select ID, rank ( ) over (order by GPA desc nulls last) as
s_rank
                        from student_grades
```