Structured Query Language (SQL)

SQL … Chapters 6 & 7 (7th edition)
Chapters 4 & 5 (6th edition)

Structured Query Language (SQL)

SQL

- **S**tructured **Q**uery **L**anguage
- major reason for the success of relational databases
- considered a declarative language
- does have some procedural elements
  - **CTE** / **Recursive queries** in 7th edition
  - Stored procedures
  - **Triggers**
- defined by the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987
- enhanced several times: SQL 86, SQL99, …

Structured Query Language (SQL)

SQL

- DDL: Define schemas, domains, tables, views, …

- DML: Retrieve and update data
  - **Select**
  - Update
  - Delete
  - Insert
- DCL: Data control language – grant access to db objects

- TCL: Transaction control language - used to manage different transactions occurring within a database…. commit, rollback

- statements end with a semicolon – not always enforced

Data Control Language - DCL

Grant …
Revoke …

```
CREATE USER   "smith-j"
      with PASSWORD '1234567ACS!'  CREATEDB ;

GRANT Select on company.* to smith-j;
```

Data Definition Language - DDL

Create database
Create table
Create view
Create domain

Alter table

Drop …

Expect a DBMS to have Create Index and Drop Index commands, but those are not part of the standard now

Schema

- `CREATE database COMPANY;`
  - A named collection of tables

CREATE TABLE

- Specify a new relation
  - Name the relation and specify attributes & constraints
  - Relation is called a *base* table (as opposed to a virtual table)

CREATE TABLE *table name* (

column definitions/constraints

table constraints

)

The CREATE TABLE

column definitions/constraints

- column name
- datatype
- Constraints
  - nulls allowed
  - not null
  - primary key
  - foreign key
  - unique
  - default
  - check

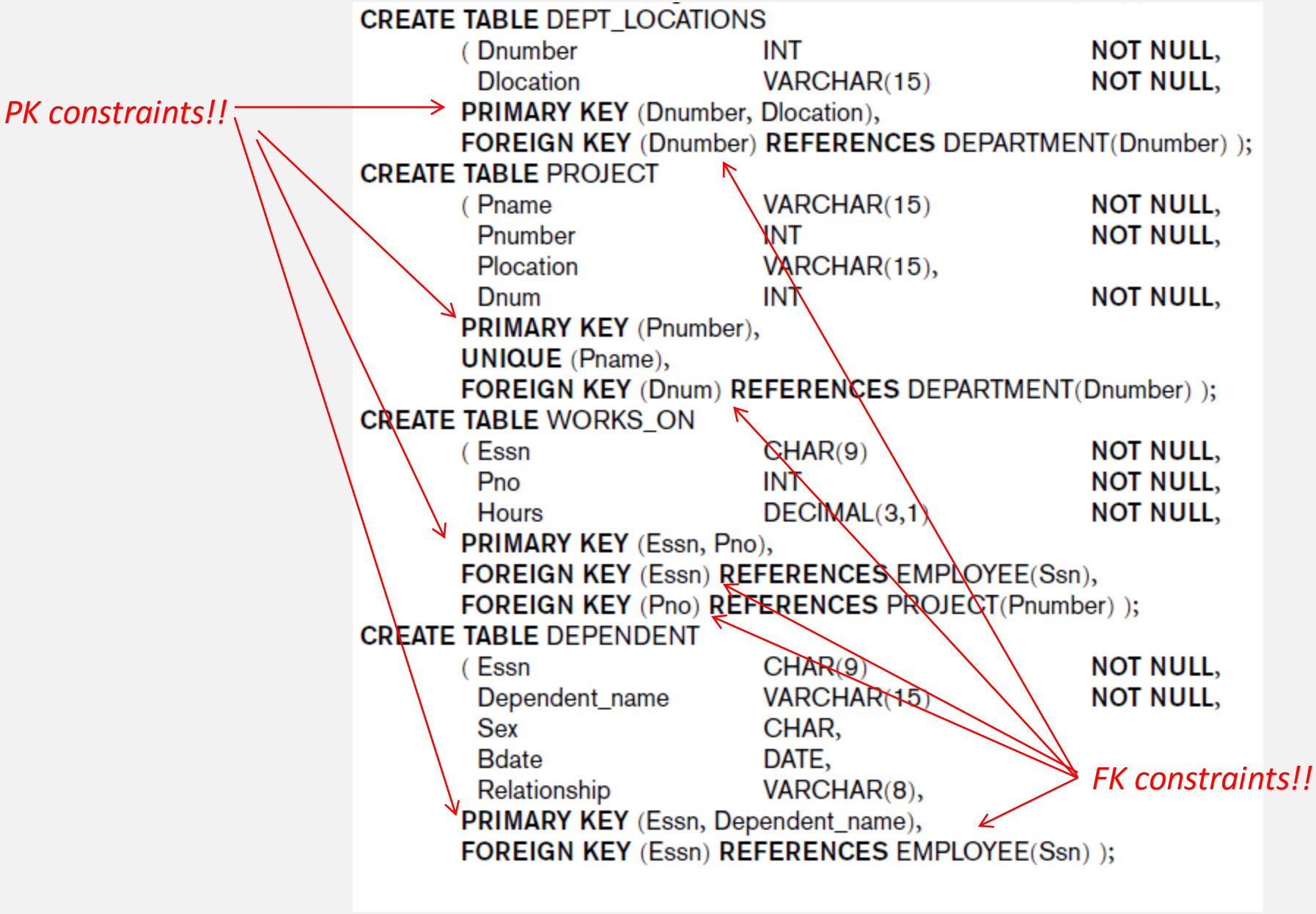
```
CREATE TABLE EMPLOYEE
        ( Fname             VARCHAR(15)          NOT NULL,
          Minit             CHAR,
          Lname             VARCHAR(15)          NOT NULL,
          Ssn               CHAR(9)              NOT NULL,
          Bdate             DATE,
          Address           VARCHAR(30),
          Sex               CHAR,
          Salary            DECIMAL(10,2),
          Super_ssn         CHAR(9),
          Dno               INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE DEPARTMENT
        ( Dname             VARCHAR(15)          NOT NULL,
          Dnumber           INT                  NOT NULL,
          Mgr_ssn           CHAR(9)              NOT NULL,
          Mgr_start_date    DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber           INT             NOT NULL,
          Dlocation         VARCHAR(15)     NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname             VARCHAR(15)     NOT NULL,
          Pnumber           INT             NOT NULL,
          Plocation         VARCHAR(15),
          Dnum              INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn              CHAR(9)         NOT NULL,
          Pno               INT             NOT NULL,
          Hours             DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn              CHAR(9)         NOT NULL,
          Dependent_name    VARCHAR(15)     NOT NULL,
          Sex               CHAR,
          Bdate             DATE,
          Relationship      VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Data types

Not null constraint!!

```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber          INT             NOT NULL,
          Dlocation        VARCHAR(15)     NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname            VARCHAR(15)     NOT NULL,
          Pnumber          INT             NOT NULL,
          Plocation        VARCHAR(15),
          Dnum             INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn             CHAR(9)         NOT NULL,
          Pno              INT             NOT NULL,
          Hours            DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn             CHAR(9)         NOT NULL,
          Dependent_name   VARCHAR(15)     NOT NULL,
          Sex              CHAR,
          Bdate            DATE,
          Relationship     VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```


PK constraints!!
CREATE TABLE DEPT_LOCATIONS
(Dnumber INT NOT NULL,
Dlocation VARCHAR(15) NOT NULL,
PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber));
CREATE TABLE PROJECT
(Pname VARCHAR(15) NOT NULL,
Pnumber INT NOT NULL,
Plocation VARCHAR(15),
Dnum INT NOT NULL,
PRIMARY KEY (Pnumber),
UNIQUE (Pname),
FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber));
CREATE TABLE WORKS_ON
(Essn CHAR(9) NOT NULL,
Pno INT NOT NULL,
Hours DECIMAL(3,1) NOT NULL,
PRIMARY KEY (Essn, Pno),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber));
CREATE TABLE DEPENDENT
(Essn CHAR(9) NOT NULL,
Dependent_name VARCHAR(15) NOT NULL,
Sex CHAR,
Bdate DATE,
Relationship VARCHAR(8),
PRIMARY KEY (Essn, Dependent_name),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn));
FK constraints!!

Attribute Data Types

No detailed knowledge of data types required for 3902 – just a working knowledge

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(`*n*`)`, `CHARACTER(`*n*`)`
    - Varying length : `VARCHAR(`*n*`)`, `CHAR VARYING(`*n*`)`, `CHARACTER VARYING(`*n*`)`

Attribute Data Types

- **Bit-string** data types
  - Fixed length: `BIT(`*n*`)`
  - Varying length: `BIT VARYING(`*n*`)`
- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`
- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD

Attribute Data Types

- **Timestamp** data type (`TIMESTAMP`)
  - Includes the `DATE` and `TIME` fields
  - Plus a minimum of six positions for decimal fractions of seconds
  - Optional `WITH TIME ZONE` qualifier
- **`INTERVAL`** data type
  - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

Domains

- Domain
  - Name to be used with attribute specifications
  - Makes it easier to change the data type for a domain that is used by numerous attributes
  - Improves schema readability

Considered really useful!!

Domains

```
CREATE DOMAIN   birthday  datetime   NULLS ALLOWED

CREATE TABLE   employee(
        id                  char(5),
        firstName       char(30),
        lastName        char(40),
        dateOfBirth     birthday
)
```

More informative

CHECK constraints

Again, onsidered really useful

- **CHECK** clause
- Specifies a *boolean expression* applied to a single row that must evaluate to true for the row to be accepted on insert or delete

```
departNo  INT NOT NULL
CHECK (departNo > 0 AND departNo < 21)
```

https://www.postgresql.org/docs/9.4/sql-createtable.html

MySQL parses the check clause but does not implement it.
https://dev.mysql.com/doc/refman/8.0/en/create-table-check-constraints.html

CHECK Constraints

- `CHECK` clauses at the end of a `CREATE TABLE` statement can specify more than one column in the row

```
CHECK  (endDate >= startDate)
```

CHECK Constraints

- To add the constraint later:

```
ALTER TABLE [dbo].products
WITH CHECK
ADD CHECK  (unitsinstock>=0);
```

Key Constraints

- A single attribute can be specified as primary key or as unique
- If multiple attributes comprise a key or if a combination of attribute values must be unique:
  - PRIMARY KEY clause
    - Specifies one or more attributes comprising the primary key of a relation
  - UNIQUE clause
    - Specifies alternate (secondary) keys

To add the constraint later: `ALTER TABLE` …

Sequences / Surrogate keys

A sequence generator is a named object that produces new values as newly inserted rows require. Can be used by more than one table.

```
CREATE SEQUENCE name
  [ INCREMENT increment_value ]
  [ MINVALUE minimum_value ]
  [ MAXVALUE maximum_value ]
```

https://www.postgresql.org/docs/9.5/static/sql-createsequence.html

Some designers want all their keys to be surrogates

ACS-4904: Data Warehousing star schema

CREATE VIEW

Creates a virtual table giving it a name and columns defined through a **Select** statement

Traditionally (*in ACS-3902*) these are **materialized on demand**, and so they are always up-to-date.

Materialized views are used in some data warehouses
→ *ACS-4904; Query rewrite.*

An SQL statement referencing a view will be *rewritten by the query processor* transparently to user so that the view definition is incorporated prior to execution.

CREATE VIEW

CREATE VIEW ***ProductQuantities***
AS
SELECT ProductID, ProductName, UnitsInStock
FROM dbo.Products

Grant Select on ***ProductQuantities*** *to …*

Can be used *to limit the information a user sees*.
Instead of granting access to Products, one can grant access to ProductQuantities.

Provides a type of security by limiting what a user sees,
and what they can do.

Can be *used to build an answer to a complex query*. A query can reference a view as if it's a table, or, a view can reference another view as if it's a table. Later: CTE

Referential Integrity

- A *foreign key* is a combination of attributes that must either be null, or have a value in an associated PK
- If a PK is composite then a FK is composite

- `FOREIGN KEY` clause
  - Relates a FK attribute to a PK attribute and can specify actions on update and delete:

    `SET NULL`

    `CASCADE`

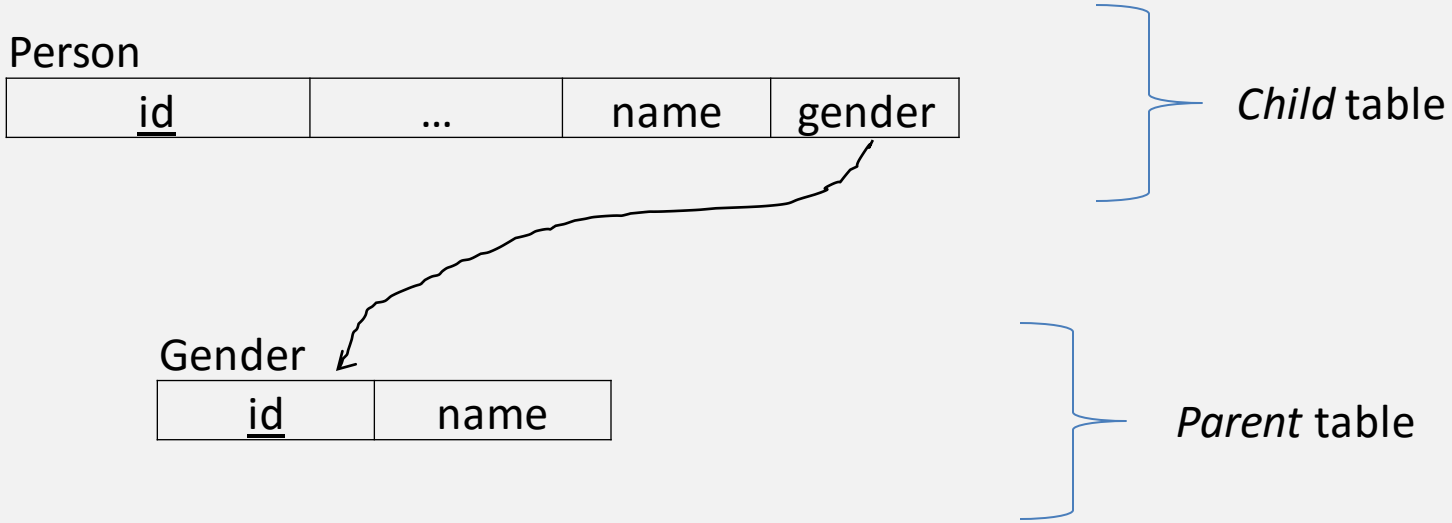
    `SET DEFAULT`

    …

REFERENCES Clause

Example:
gender in Person is a foreign key referencing the PK id in Gender

The action clause says what happens for delete or update of a PK value



The *action clause* specifies what should happen in a *child* table
if one deletes a row in the *parent* table

REFERENCES Clause – creating the Person, Gender tables

```
create table gender (
    id          int    primary key,
    name        varchar(25)   not null unique
    );

create table person (
    id          int primary key,
    name        varchar(25),
    gender      int
    );
```

→ Next slide

REFERENCES Clause – creating the Person, Gender tables

```
ALTER TABLE person add constraint persongender foreign
key (gender) references gender(id)
on delete …
on update … ;
```

Specify actions when a row is deleted/updated in referenced table

```
INSERT INTO gender  VALUES (1, 'FEMALE');
INSERT INTO gender  VALUES (2, 'MALE');

INSERT INTO person  VALUES (1, 'JOE', 1);
INSERT INTO person  VALUES (2, 'TOM', 1);
```

REFERENCES Clause

Delete from Gender where id=1;

NO ACTION
If a row of Employee has a gender id value of 1 the delete is disallowed (and is an error!)

CASCADE
If a row of Employee has a gender id value of 1 then the Employee row is deleted too
Deletes can be propagated.

SET NULL
If a row of Employee has a gender id value of 1 the FK is set to null

SET DEFAULT
If a row of Employee has a gender id value of 1 the FK is set to its default value.

REFERENCES Clause

Update Gender
set genderId=99
where id=1;

NO ACTION
If a row of Employee has a gender id value of 1 the update is disallowed

CASCADE
If a row of Employee has a gender id value of 1 then the Employee row's FK is updated to 99
Updates can be propagated.

SET NULL
If a row of Employee has a gender id value of 1 the FK is set to null

SET DEFAULT
If a row of Employee has a gender id value of 1 the FK is set to its default value.

DML – data maniputlation language

Select

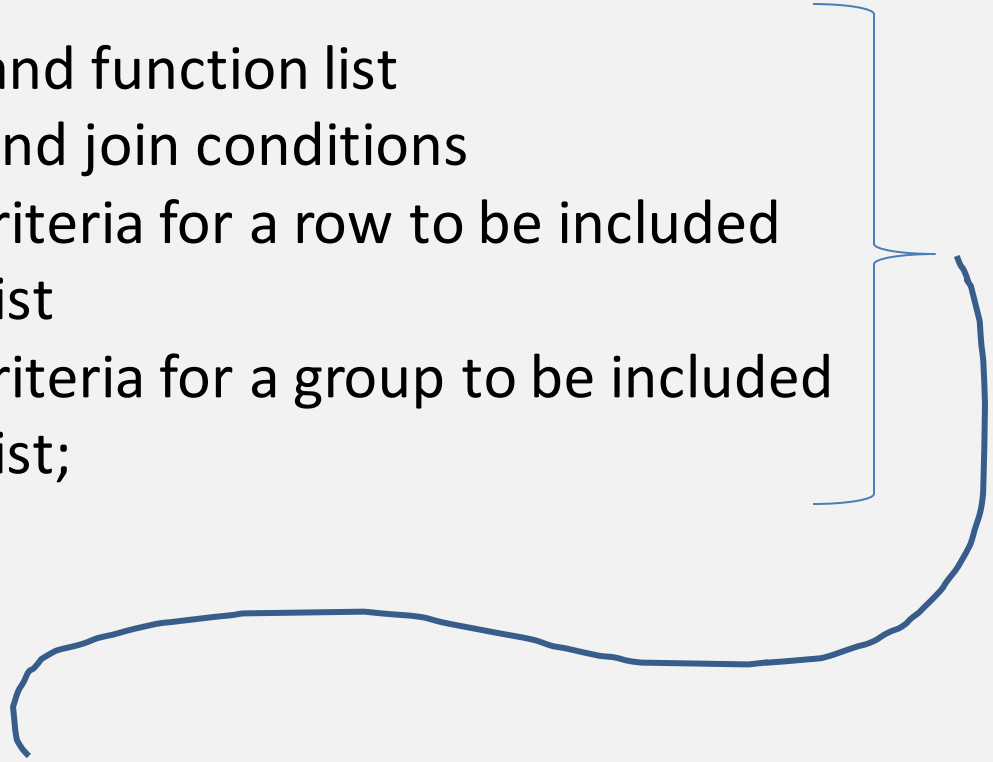
Update

Insert

Delete

SQL Select – general syntax

Select	attribute and function list
From	table list and join conditions
Where	boolean criteria for a row to be included
Group by	attribute list
Having	boolean criteria for a group to be included
Order by	attribute list;



Text goes through this in parts

SQL Select – execution

The database optimizer determines the exact nature of how and when rows are retrieved and processed.

Conceptual execution sequence:

1. Rows are retrieved from the referenced tables (cartesian product)
2. Rows that match the *where* criteria are retained
3. Rows are joined based on *join* criteria
4. Rows are grouped into non-overlapping groups according to values of the *grouping* attributes
5. Groups that match the *having* criteria are retained
6. From the rows (or groups) left they are presented to the user in sequence according to the *order by* attribute values

Results of SQL queries when applied to the COMPANY

(a)

Bdate	Address
1965-01-09	731Fondren, Houston, TX

(b)

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

Results of SQL queries when applied to the COMPANY

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

Case expressions

The case/when/then/else/end expression, which was introduced in SQL-92.

```
CASE WHEN n > 0
        THEN 'positive'
    WHEN n < 0
        THEN 'negative'
    ELSE 'zero'
END
```

```
CASE n WHEN 1
        THEN 'one'
     WHEN 2
        THEN 'two'
     ELSE 'i cannot count that high'
END
```

```
SELECT [ProductID],[ProductName],
            CASE WHEN unitsinstock>100 THEN 'HIGH' ELSE 'low' END
  FROM [3902northwind].[dbo].[Products]
```

JOINS

Whenever we use a query to retrieve data from two or more tables, the database query processor performs an operation called a join.

Primary classification for joins

- inner join
- natural join
- outer join: right, left, full
- Cartesian product (or cross product)

Some other classifications

- equi-join
- non-equi join
- self-join
- anti-join

Inner join

Select … from A inner join B on(…)

A row of A and a row of B are joined to form one row for the result when they match according to the ON condition

```
Select *
from orders inner join customers
on (orders.customerid = customers.customerid)
```

Note an inner join can be specified in the *where* clause, but this is not recommended.

```
Select *
from orders, customers
where orders.customerid = customers.customerid
```

Inner join

Recode Q1, Q2, Q8 as inner joins

Natural join

A NATURAL join on two relations R and S has no join condition specified and is an implicit EQUIJOIN for each pair of attributes with same name from R and S

Select … from A inner join B on(…)

A row of A and a row of B are joined to form one row for the result when they match according to the ON condition

Select orderID, contactName
from orders natural join customers ;

Natural join

Page 4 Q ‘1B’ is a natural join

Left Outer join

Select … from A left outer join B on(…)

A row of A and a row of B are joined to form one row for the result when they match according to the ON condition.
If a row of A does not match a row of B then A is joined to a null row.

e.g. to list each employee and their dependents:

Right outer join – left to student

Left Outer join

Page 4 Q 8B is a left outer join

Recode this as a right outer join

Full Outer join

Select … from A full outer join B on(…)

A row of A and a row of B are joined to form one row for the result when they match according to the ON condition.
If a row of A does not match a row of B then A is joined to a null row.
If a row of B does not match a row of A then B is joined to a null row.

Not in MySQL
In PostgreSQL

Cartesian Product / Cross Product

Select … from A, B ;

There is no join criteria and so each row of A joins to each row of B to form one row for the result.

The number of rows in the result can be enormous.

Cartesian Product / Cross Product

P 1 Q10 lists all combinations of ssn and dept name

How would we list all combinations of first and last names?

Other Joins

equi-join
the ON condition uses the= operator

non-equi join
The ON condition uses any operator other than =

e.g. list each employee and anyone who earns more than some specific employee

Other Joins

self-join
A table is joined to itself.

P1 Q8 is a self join

e.g. find the employees who are older than Alicia Zelaya (of course, without being aware Alicia was born on jan 19, 1968) . This query is also an example of a non-equi join.

e.g. list each employee and the name of their supervisor

e.g. like the above, but your list must include all employees

anti-join

Given an ON condition for joining A and B, list the rows of A that do not join to a row in B.

e.g. list the employees who are not supervising anyone

e.g. in the assignment 1 database list any staff member who is not teaching any courses.

Ambiguous Attribute Names

- If two attributes of the same name are specified in a SQL statement then they must be qualified with the table name using dot notation – makes references unambiguous

```
Q1A:    SELECT    Fname, EMPLOYEE.Name, Address
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DEPARTMENT.Name='Research' AND
                  DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Assuming Lname in Employee is replaced by Name
Dno in Employee is replaced by Dnumber

Ambiguous Attribute Names

- Consider listing each department name and its locations
  - Need to look at the Department and Dept_Locations tables
  - Both have an attribute Dnumber… any reference to this attribute must be unambiguous

Aliases & Renaming

- An alias can be given for a table
- Attributes can be renamed (not in MySQL)

```
Select fn, ln, bal
from Owners o (ono, fn, ln)
inner join
Accounts a (acct, ono, bal)
on (o.ono=a.ono)
where bal > 1000 ;
```

Operator	Description	Example
=	Equal to	`Author = 'Alcott'`
<>	Not equal to (most DBMS also accept != instead of <>)	`Dept <> 'Sales'`
>	Greater than	`Hire_Date > '2012-01-31'`
<	Less than	`Bonus < 50000.00`
>=	Greater than or equal	`Dependants >= 2`
<=	Less than or equal	`Rate <= 0.05`
`BETWEEN`	Between an inclusive range	`Cost BETWEEN 100.00 AND 500.00`
`LIKE`	Match a character pattern	`First_Name LIKE 'Will%'`
`IN`	Equal to one of multiple possible values	`DeptCode IN (101, 103, 209)`
`IS` *or* `IS NOT`	Compare to null (missing data)	`Address IS NOT NULL`
`IS NOT DISTINCT FROM`	Is equal to value or both are nulls (missing data)	`Debt IS NOT DISTINCT FROM`
`AS`	Used to change a field name when viewing results	`SELECT employee AS 'department1'`

Ch 4 Basic SQL

From postgreSQL documentation for
Is distinct from
Is not distinct from

Truth Table

```
\pset null '<<NULL>>'

select a.a, b.b, a.a IS DISTINCT FROM b.b AS "Is Distinct From" FROM (VALUES (1), (2), (NULL)) AS a (a), (VALUES (1), (2), (NULL)) AS b (b);
```

a	b	Is Distinct From
1	1	f
1	2	t
1	<<NULL>>	t
2	1	t
2	2	f
2	<<NULL>>	t
<<NULL>>	1	t
<<NULL>>	2	t
<<NULL>>	<<NULL>>	f

```
select a.a, b.b, a.a IS NOT DISTINCT FROM b.b AS "Is Not Distinct From" FROM (VALUES (1), (2), (NULL)) AS a (a), (VALUES (1), (2), (NULL)) AS b (b);
```

a	b	Is Not Distinct From
1	1	t
1	2	f
1	<<NULL>>	f
2	1	f
2	2	t
2	<<NULL>>	f
<<NULL>>	1	f
<<NULL>>	2	f
<<NULL>>	<<NULL>>	t

Use of distinct

- SQL does not automatically eliminate duplicate tuples in query results
- Use the keyword **`DISTINCT`** in the `SELECT` clause causes duplicates to be eliminated from result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

ALL is the default and is optional to include

```
Q11:   SELECT   ALL Salary
       FROM     EMPLOYEE;

Q11A:  SELECT   DISTINCT Salary
       FROM     EMPLOYEE;
```

Use of distinct

- P 1 Q 11A list all salaries with no duplicates
- P 2 Q 4A
- P 4 Q 17
- P 5 Q 23

Set operations

- `UNION`, `EXCEPT` (difference), `INTERSECT`

What are:

```
A union B
A except B
A intersect B
```

Not in MySQL (`A except B`)

in PostgreSQL (`A union B`, `A except B`, `A intersect B`)

Queries
P2 Q 4A

Set operations

List all first names in company database

List first names of employees that are not first names of any depenedents

Tables as Sets in SQL

- Set operations
  - Union, except, intersect eliminate duplicates
  - Equivalent multiset operations (no duplicate elimination)

    `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL`)

Substring Pattern Matching

- **`LIKE`** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character

  E.g.

  P 2 Q12 list all employees that live in Texas

  P 2 Q12A list all employees born in the 50s

IN operator

- **`IN`** can be used with a list (*or with a subquery – later*)
- Syntax: *an attribute* IN (*a comma-separated list of values*)

  evaluates to true if the attribute's value exists in the comma-separated list

  E.g.

  P4 Q 17 employees working on projects 1, 2, or 3

  list all employees with first name James or Alicia

Arithmetic Operators

- Standard arithmetic operators:
  - Addition (+)
  - subtraction (–)
  - multiplication (*)
  - division (/)

E.g. list the total cost of order detail lines:

Select unitPrice*quantity*(1-discount) from [order details]

BETWEEN Operator

- P 2 Q14 all employees earning between $30,000 and $40,000

- To get all employees born in 1960s

  SELECT * FROM company.employee where bdate between '1960-01-01' and '1969-12-31';

  - Inclusive … includes the endpoints

Ordering of Query Results

- Use `ORDER BY` clause
  - Keyword `DESC` to see result in a descending order of values
  - Keyword `ASC` to specify ascending order explicitly

E.g. to list all employees in ascending order by last name and where they have the same last name in descending sequence by first name

```
Select * from employee
Order by lastName ASC, firstName DESC;
```

Ordering of Query Results

- Use `ORDER BY` clause

E.g. to list employee last names in ascending order by salary

```
Select lname from employee
Order by salary;
```

Not in select list … but okay

Ordering of Query Results

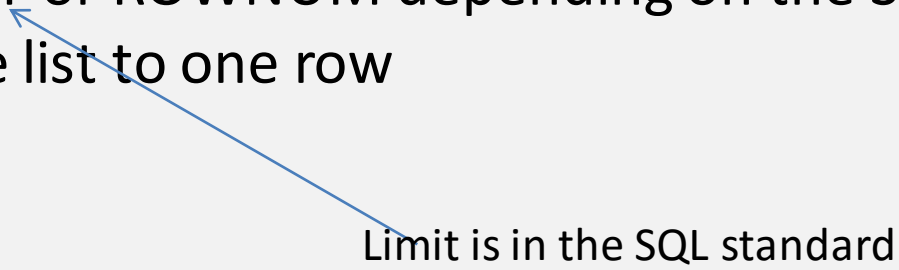
- P 2 Q 15
- List employees …

  In order by department and within department by last name then first name

Not in select list … but okay

Limiting the number of rows returned by a query

- Suppose we want to know who has the greatest salary.
- To do this we can order the results in descending order and then, using TOP or LIMIT or ROWNUM depending on the SQL dialect, we can limit the list to one row




```
SELECT * FROM company.employee
order by salary desc limit 1;
```

NULLs

- Meanings of NULL
  - Unknown value
  - Unavailable or withheld value
  - Not applicable attribute
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN

and, or, not

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

and, or, not – from PostgreSQL documentation

The usual logical operators are available:

AND
OR
NOT

SQL uses a three-valued logic system with true, false, and `null`, which represents "unknown".

a	*b*	*a* **AND** *b*	*a* **OR** *b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

a	**NOT** *a*
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Checking for NULLs

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS NULL` *or* `IS NOT NULL`

  `P3 Q18 … Employees with no supervisor`

To get orders that have not shipped:
Select * from orders where shippedDate is null ;

To get orders that have shipped:
Select * from orders where shippedDate is not null ;

Subqueries / Nested Queries

- **Nested queries**
  - A query can appear
  - In the select clause if it returns a single value
  - In the from clause as a derived table
  - In the where clause with IN, EXISTS, NOT EXISTS, UNIQUE …

  P4 Q4A … list projects where Smith is a manager of the project's department or where Smith works on the project

Correlated vs non-correlated Queries

Correlated subquery:

- Subquery references an attribute from an outer query
- Subquery must be evaluated once for each tuple in the outer query

P3 Q16 … list an employee who has a dependent of same name and sex

Correlated vs non-correlated Queries

Non-correlated

- Independent of outer query
- Can be considered to be executed first and then its result used in its place in the query

- Example. List employees who have dependents

Subqueries / Nested Queries

Can appear in the select clause if it returns a single value

E.g. List each employee and the number of children they have

```
select fname, lname, (select count(*) from dependent d
where  e.ssn = d.essn) as numberOfDependents
from employee e
```

Subqueries / Nested Queries

Can be in the select clause if it returns a single value

For each employee list the number of others earning more money.

```
select fname, lName,
        (select count(*) from employee e2
       where e1.salary < e2.salary) as NumberHigher
  from employee e1
```

Inline Views – another term for a subquery in the From clause

- In-line view

  Defined in the `FROM` clause of an SQL query

  E.g. list the number of orders from customers who are located in the US

```
Select count(*)
from orders o inner join
                        (
                        Select customerID
                        from customers
                        where country='U.S."
                        ) c
on (c.customerID = o.customerID) ;
```

Subqueries / Nested Queries

In the From clause

list employees working on projects in Houston

```
SELECT essn from works_on w
  inner join
  (select pnumber from project where plocation = 'Houston')  as x
  on x.pnumber = w.pno
```

Subqueries / Nested Queries in Where clause

e.g. list employees earning more than the average salary

```
Select lastname from Employee
 where salary > (        select avg(salary)
                         from Employee
                         )
```

Subqueries / Nested Queries

- In Where clause … the `IN` operator
  - Evaluates to `TRUE` if comparison value(s) is/are one of the elements produced by subquery

Nested Queries

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

Nested Queries

- Use tuples of values in comparisons
  - Place them within parentheses

```
SELECT essn
from works_on
where (pno, Hours) in
(select pno, hours from works_on
  where essn = '123456789');
```

Nested Queries

- comparison operators

  =, >, >=, <, <=, <>

  ANY

  SOME

  ALL

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL ( SELECT   Salary
                        FROM     EMPLOYEE
                        WHERE    Dno=5 );
```

EXISTS and NOT EXISTS

- `EXISTS`
  - Evaluates `true` if the result of a correlated nested query is not empty

  E.g. list any employee who has a dependent with the same first name
  P3 Query 16B.

The EXISTS and UNIQUE Functions in SQL

- `UNIQUE`
  - Evaluates `TRUE` if there are no duplicate tuples in the result of some subquery

E.g. list those employees with one dependent

```
Select *
        from employee c
        Where unique (
                        select essn
                        from dependent d inner join employee o
                        on (o.ssn = d.essn)
                        where o.ssn=c.ssn)
```

UNIQUE used this way is not in PostgreSQL
… could use … `where 1 = (select count(*) from … as xx )`

Aggregate Functions

- Used to summarize information from multiple tuples into a single-tuple summary
- **Grouping**
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- Functions can be used in the `SELECT` clause or in a `HAVING` clause
- If no Group By then aggregates are applied to all rows that satisfy the where clause

Aggregate Functions

- OK:

  select count(ssn) from employee;

- Not OK

  select fname, count(ssn) from employee;

  But could have

  select fname, (select count(ssn) from employee) from employee;

Aggregate Functions

- NULL values discarded when aggregate functions are applied to a particular column

p5 Q20

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```

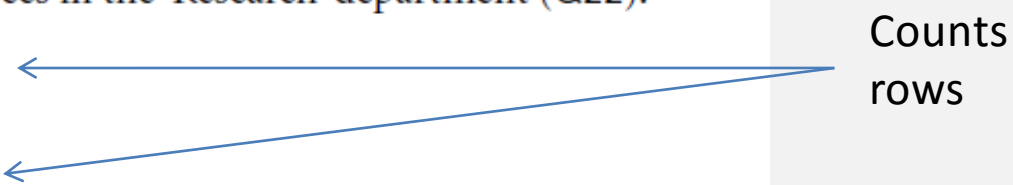
Aggregate Functions

Count(*) counts rows in a result set

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT  COUNT (*)
      FROM    EMPLOYEE;

Q22:  SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND DNAME='Research';
```



GROUP BY and HAVING Clauses

- `GROUP BY` clause
  - Specifies grouping attributes
  - Values are used to partition rows into subsets
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute
- HAVING clause
  - Specifies a boolean condition pertinent to a group. If not satisfied by a group then the group is eliminated from the result set

GROUP BY and HAVING Clauses

E.g. P5 Q24

Number of employees and average salary for each department

For each employee list the number of dependents

a) For only those with depedents

b) For all employees

GROUP BY and HAVING Clauses

E.g. list employees on at least four projects

```
Select ssn, fname, count(*)
from employee inner join works_on on (ssn = essn)
group by ssn, fname
having count(*)  > = 4
```

Must be single-valued for a group

The criteria a group must satisfy to be included in the result set