Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology

Chapter 4: The SQL Language

Database Systems (CO2013)

Computer Science Program

Assoc. Prof. Dr. Võ Thị Ngọc Châu

(chauvtn@hcmut.edu.vn)

Semester 1 – 2022-2023

Content

- Chapter 1: An Overview of Database Systems
- Chapter 2: The Entity-Relationship Model
- Chapter 3: The Relational Data Model
- **Chapter 4: The SQL Language**
- Chapter 5: Relational Database Design
- Chapter 6: Physical Storage and Data Management
- Chapter 7: Database Security

Chapter 4: The SQL Language

- 4.1. Introduction to the SQL language
- 4.2. DDL
- 4.3. DML
- 4.4. DCL
- 4.5. Stored Functions
- 4.6. Stored Procedures
- 4.7. Triggers

Main References

Text:

- [1] R. Elmasri, S. R. Navathe, Fundamentals of Database Systems- 6th Edition, Pearson- Addison Wesley, 2011.
  - ***R. Elmasri, S. R. Navathe, Fundamentals of Database Systems- 7th Edition, Pearson, 2016.***

References:

[1] S. Chittayasothorn, *Relational Database Systems: Language, Conceptual Modeling and Design for Engineers*, Nutcha Printing Co. Ltd, 2017.

[3] A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts – 7th Edition, McGraw-Hill, 2020.

[4] H. G. Molina, J. D. Ullman, J. Widom, *Database Systems: The Complete Book - 2nd Edition,* Prentice-Hall, 2009.

[5] R. Ramakrishnan, J. Gehrke, *Database Management Systems – 4th Edition,* McGraw-Hill, 2018.

[6] M. P. Papazoglou, S. Spaccapietra, Z. Tari, *Advances in Object-Oriented Data Modeling*, MIT Press, 2000.

[7]. G. Simsion, *Data Modeling: Theory and Practice*, Technics Publications, LLC, 2007.

Introduction to the SQL language

SQL (Structured Query Language)	
Paradigm	Declarative
Family	Query language
Designed by	Donald D. Chamberlin Raymond F. Boyce
Developer	ISO/IEC
First appeared	1974; 46 years ago
Stable release	SQL:2016 / December 2016; 3 years ago
Typing discipline	Static, strong
OS	Cross-platform
Website	www.iso.org/standard/63555.html

Major implementations
Many
Dialects
SQL-86 • SQL-89 • SQL-92 • SQL:1999 • SQL:2003 • SQL:2006 • SQL:2008 • SQL:2011 • SQL:2016
Influenced by
Datalog
Influenced
CQL, LINQ, SPARQL, SOQL, PowerShell,[1] JPQL, jOOQ, N1QL

SQL (file format)	
Filename extension	`.sql`
Internet media type	`application/sql`[2][3]
Developed by	ISO/IEC
Initial release	1986
Type of format	Database
Standard	ISO/IEC 9075
Open format?	Yes

SQL – Wikipedia, Accessed 28/09/2020

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types (arrays), and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT).
2003	SQL:2003		Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[34]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[35] FETCH clause.
2011	SQL:2011		Adds temporal data (PERIOD FOR)[36] (more information at: Temporal database#History). Enhancements for window functions and FETCH clause.[37]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON.
2019	SQL:2019		Adds Part 15, multidimensional arrays (MDarray type and operators).

SQL – Wikipedia, Accessed 28/09/2020

Introduction to the SQL language

- SQL = Structured *Query* Language
  - A standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
  - One of the first commercial languages to utilize the relational data model
- A declarative *non-procedural* language (4GL) supported by existing RDBMSs
  - Based on tuple relational calculus, associated with relational algebra for *internal* representation, processing, and optimization
  - Specify "WHAT" in data requests on a database
  - *NOT* specify "*HOW*" to process the data requests

SQL vs. The Relational Model

SQL	The Relational Model
Table (set if a primary key is specified)	Relation (set)
Column	Attribute (+ *attribute values*)
Row	Tuple
Domain	Domain
Data type	(*Data type*)
Primary key	Primary key
Uniqueness, NOT NULL	Secondary key
Foreign key	Foreign key
Defined and enforced with CHECK, ASSERTION, TRIGGER	Semantic constraints

SQL vs. The Relational Algebra

SQL	The Relational Algebra
SELECT statement - specifies WHAT	Relational algebra expression - specifies WHAT + HOW
FROM clause	(*none*), x, *, other joins
WHERE clause	SELECTION (σ)
SELECT clause (+ DISTINCT)	PROJECTION (π)
Aggregate functions	Aggregate functions
GROUP BY clause	<grouping attributes>$\Im$
HAVING clause	SELECTION on $\Im$
ORDER BY clause	(*not available*)
UNION, INTERSECT, MINUS	$\cup$, $\cap$, -
INSERT, DELETE, UPDATE statements	(*not available*)

(*DBMS-specific implementation*)

SQL Data Types

- Predefined data types
  - Character Types: Character fixed (CHAR) (e.g. CHAR(10)), Character Varying (VARCHAR) (e.g. VARCHAR(20)), Character Large Object (CLOB)
  - Binary Types: Binary (BINARY), Binary Varying (VARBINARY), Binary Large Object (BLOB)
  - Numeric Types
    - Exact Numeric Types (NUMERIC, DECIMAL, SMALLINT, INTEGER, BIGINT)
    - Approximate Numeric Types (FLOAT, REAL, DOUBLE PRECISION)
  - Datetime Types (DATE, TIME, TIMESTAMP)
  - Interval Type (INTERVAL)
  - Boolean (three-valued logic (3VL): true, false, *unknown* (NULL))
  - XML, JSON
- Constructed types
  - ARRAY, MULTISET, REF(erence), or ROW
- User-defined types
  - comparable to classes in object-oriented language with their own constructors, observers, mutators, methods, inheritance, overloading, overwriting, interfaces, …

SQL Data Types – 3-Valued Logic

AND	True (T)	False (F)	Unknown (U)
True	T	F	U
False	F	F	F
Unknown	U	F	U

NOT	X
True	F
False	T
Unknown	U

OR	True	False	Unknown
True	T	T	T
False	T	F	U
Unknown	T	U	U

Unknown ≈ NULL

X: True,
False,
Unknown

Introduction to the SQL language

- Data Definition Language (DDL)
  - CREATE, ALTER, DROP
- Data Manipulation Language (DML)
  - For *queries*: SELECT
  - For data *modifications*: INSERT, DELETE, UPDATE
- Data Control Language (DCL)
  - For *access* control: GRANT, REVOKE
  - For *transaction* control: COMMIT, ROLLBACK, SET AUTOCOMMIT OFF

→ SQL: *case-insensitive*, *extended* in DBMSs

SQL

Simplified SQL Statements (*Checked with each DBMS*)

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                           { , <column name> <column type> [ <attribute constraint> ] }
                           [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
```

```
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                       { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

```
DELETE FROM <table name>
[ WHERE <selection condition> ]
```

```
UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]
```

```
CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]
```

```
DROP INDEX <index name>
```

```
CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>
```

```
DROP VIEW <view name>
```

Source: [1]

Data Definition Language

- Purpose
  - Create, modify, and remove the constructs at the structure level of a database
- Named constructs
  - Database (schema), tables, types, domains, constraints, indexes, views, assertions, triggers, …
- DDL statements
  - CREATE
  - DROP
  - ALTER

CREATE SCHEMA

- Starting with SQL2, a *schema* is defined to group together tables and other constructs that belong to the *same* database application.
  - tables, types, constraints, views, domains, indexes, authorization grants, etc.
- A *catalog* is a named collection of schemas.
  - Database installations typically have a default environment and schema, so when a user connects and logs in to that database installation, the user can refer directly to tables and other constructs within that schema without having to specify a particular schema name.
  - In a catalog, a special schema is INFORMATION_SCHEMA, which provides information on all the schemas in the catalog and all the element descriptors in these schemas.
  - Schemas within the same catalog can also share certain elements, such as type and domain definitions.

CREATE SCHEMA

DBMS-dependent
- MySQL
+ PostgreSQL
...

- An SQL schema is identified by a *schema name* and includes an *authorization identifier* to indicate the user or account who owns the schema, and descriptors for each element.
- The *privilege* to create schemas, tables, and other constructs must be explicitly granted to the relevant *user* accounts by the system administrator or DBA.

CREATE SCHEMA *SchemaName*
[AUTHORIZATION *AuthorizationIdentifier*];

CREATE SCHEMA *Company* AUTHORIZATION *Smith*;

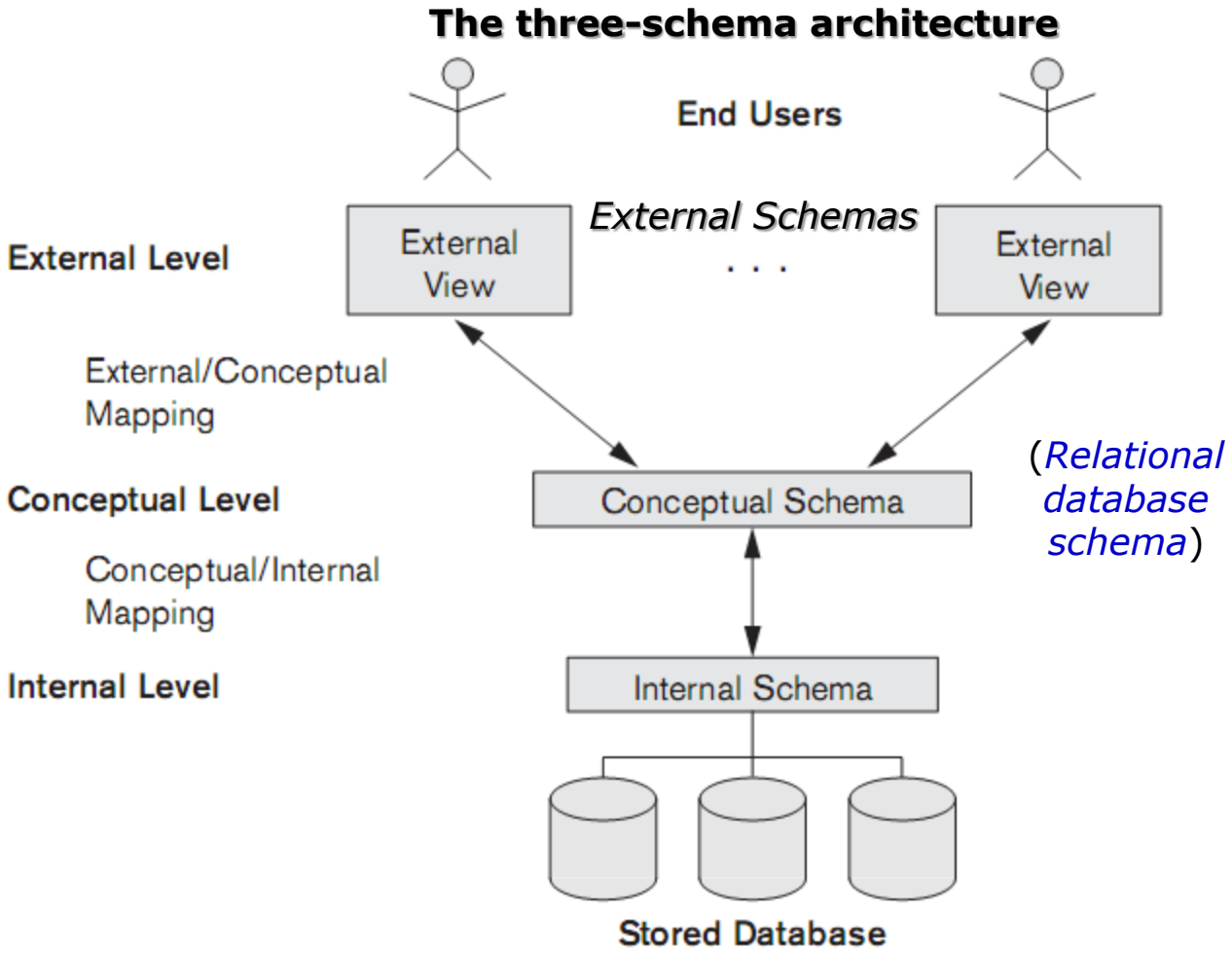
CREATE SCHEMA *Company*;

NOTES on SCHEMAS

the part of the database that a particular user group is interested in and hides the rest of the database from that user group

the structure of the whole database for a community of users

the physical storage structure of the database



- *Conceptual schema in the three-schema architecture ≠ the conceptual EER schema* in design!!!
- Conceptual schema in the three-schema architecture is in fact *the relational database schema*.
- *External Schemas* are supported with CREATE VIEW.

NOTES on SCHEMAS

- MySQL
  - CREATE SCHEMA = CREATE DATABASE
  - Concepts are equivalent to the standard.
- MS SQL Server
  - CREATE DATABASE with more physical properties
    - Data/ log files
  - dbo (the default schema of the database owner) is a default schema for each database to group objects of similar scope or ownership.
- Oracle
  - CREATE DATABASE with more physical properties
    - Data/ log files
  - CREATE USER with a schema of the same name created by default
    - A user is an account through which we can log in to the database.
  - CREATE SCHEMA does not actually create a schema. Oracle Database automatically creates a schema when a user is created. This statement populates the schema (*the one with the username*) with tables and views and grant privileges on those objects without having to issue multiple SQL statements in multiple transactions.

CREATE TABLE

- A base table is created and actually stored as a file by a DBMS with CREATE TABLE.
  - The attributes in a base table are considered to be ordered in the sequence in which they are specified in the CREATE TABLE statement.
  - Rows (tuples) are not considered to be ordered within a table (relation).
- A virtual table is created but may or may not correspond to an actual physical file with CREATE VIEW.

CREATE TABLE

CREATE TABLE [SchemaName.]TableName

({columnName dataType [NOT NULL] [UNIQUE] [PRIMARY KEY]

[DEFAULT <value>]

[CHECK searchCondition] [,...]}

[PRIMARY KEY (listOfColumns)] [,…]

{[UNIQUE (listOfColumns)] [,…]}

{[FOREIGN KEY (listOfFKColumns)

REFERENCES [SchemaName.] ParentTableName [(listOfPKColumns)]

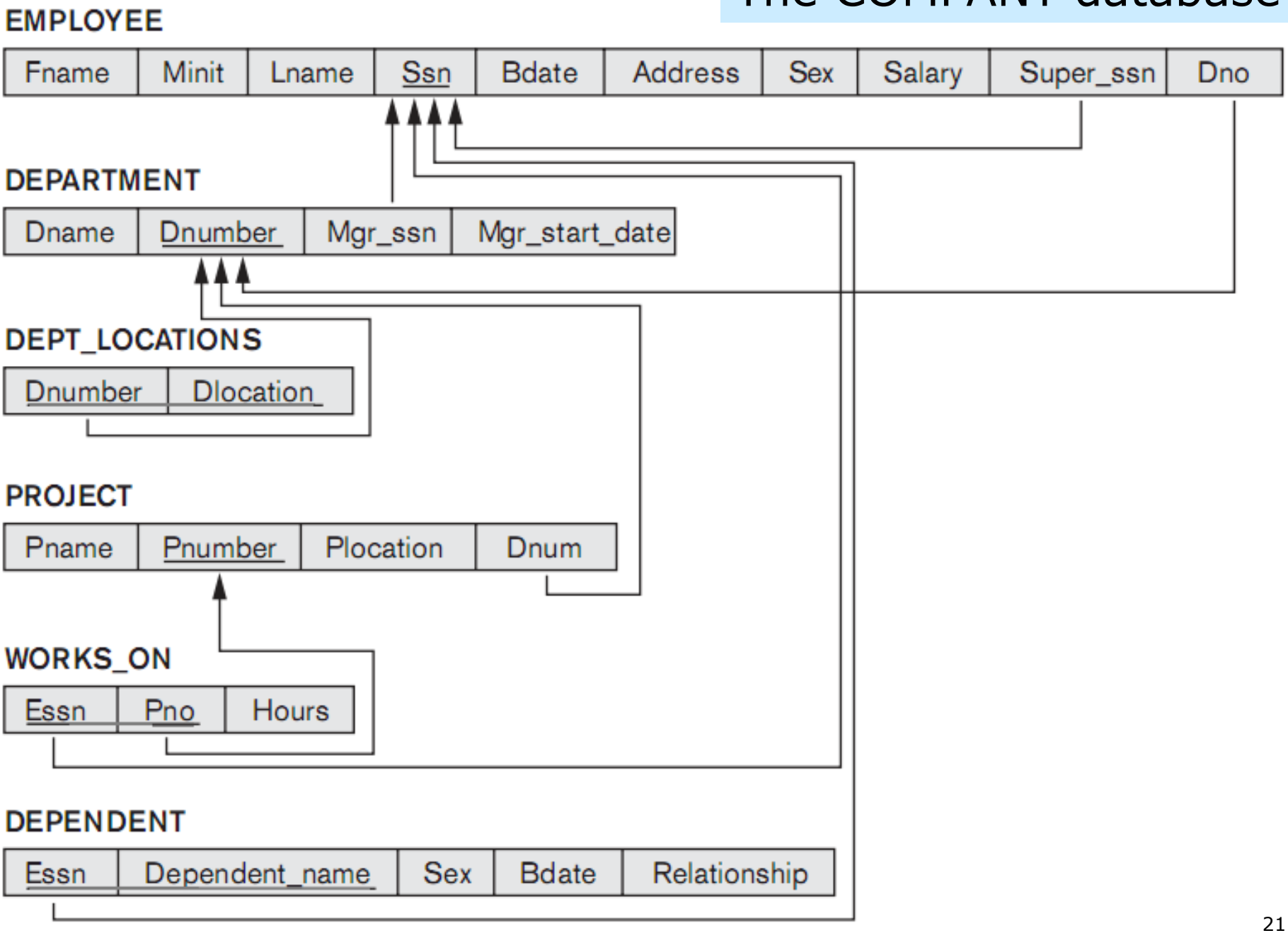
[ON UPDATE CASCADE | SET NULL]

[ON DELETE CASCADE | SET NULL] [,…]}

{[CHECK (searchCondition)] [,…] }

{[CONSTRAINT constraintName PRIMARY KEY| UNIQUE| FOREIGN KEY| CHECK constraint…] [,…]})

The COMPANY database



```
CREATE SCHEMA COMPANY;

CREATE TABLE EMPLOYEE (
        FNAME           VARCHAR(15)     NOT NULL,
        MINIT           CHAR,
        LNAME           VARCHAR(15)     NOT NULL,
        SSN             CHAR(9),
        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)
);

CREATE TABLE DEPENDENT (
        ESSN            CHAR(9),
        DEPENDENT_NAME  VARCHAR(15),
        SEX             CHAR,
        BDATE           DATE,
        RELATIONSHIP    VARCHAR(8),
        PRIMARY KEY (ESSN, DEPENDENT_NAME),
        FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN)
);
```

```
CREATE TABLE DEPARTMENT (
        DNAME           VARCHAR(15)     NOT NULL,
        DNUMBER         INT,
        MGR_SSN         CHAR(9)         NOT NULL,
        MGR_STAR_TDATE  DATE,
        PRIMARY KEY (DNUMBER),
        UNIQUE (DNAME),
        FOREIGN KEY (MGR_SSN) REFERENCES EMPLOYEE (SSN)
);

CREATE TABLE DEPT_LOCATIONS (
        DNUMBER         INT,
        DLOCATION       VARCHAR(15),
        PRIMARY KEY (DNUMBER, DLOCATION),
        FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)
);

ALTER TABLE EMPLOYEE
ADD FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNUMBER);
```

```
CREATE TABLE PROJECT (
        PNAME           VARCHAR(15)     NOT NULL,
        PNUMBER         INT,
        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),
        PNO     INT,
        HOURS   DECIMAL(3,1),
        PRIMARY KEY (ESSN, PNO),
        FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN),
        FOREIGN KEY (PNO) REFERENCES PROJECT (PNUMBER)
);
```

CREATE TABLE - CONSTRAINTS

- Basic constraints are specified as part of table creation.
  - Primary key
  - Foreign key (Referential integrity constraint)
  - UNIQUE
  - NOT NULL
  - Attribute defaults
  - Restrictions on attribute domains and other constraints on individual tuples within a table using the CHECK clause

CREATE TABLE - CONSTRAINTS

- Primary key: specified *only one* for each table

```
CREATE TABLE DEPARTMENT (
        DNAME                   VARCHAR(15),
        DNUMBER                 INT             PRIMARY KEY,
        MGRSSN                  CHAR(9),
        MGRSTARTDATE            DATE
);
CREATE TABLE WORKS_ON (
        ESSN            CHAR(9),
        PNO             INT,
        HOURS           DECIMAL(3,1),
        PRIMARY KEY (ESSN, PNO)
);
```

CREATE TABLE - CONSTRAINTS

- Foreign key: specified for *referential integrity*

```
CREATE TABLE DEPARTMENT (
        DNAME                   VARCHAR(15),
        DNUMBER                 INT             PRIMARY KEY,
        MGRSSN                  CHAR(9),
        MGRSTARTDATE            DATE
);
CREATE TABLE DEPT_LOCATIONS (
        DNUMBER       INT,
        DLOCATION     VARCHAR(15),
        PRIMARY KEY (DNUMBER, DLOCATION),
        FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)
);
```

CREATE TABLE - CONSTRAINTS

- Foreign key: specified for *referential integrity*
  - The default action for an integrity violation is to *reject* the *update* operation that will cause a violation: RESTRICT.
  - Other actions: SET NULL, CASCADE, and SET DEFAULT
    - CASCADE ON DELETE is to delete all the referencing tuples, whereas CASCADE ON UPDATE is to change the value of the referencing foreign key attribute(s) to the updated (new) primary key value for all the referencing tuples.

```
CREATE TABLE DEPT_LOCATIONS (
        DNUMBER       INT,
        DLOCATION     VARCHAR(15),
        PRIMARY KEY (DNUMBER, DLOCATION),
        FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)
                    ON DELETE CASCADE ON UPDATE CASCADE
);
```

CREATE TABLE - CONSTRAINTS

- UNIQUE: no duplicate values for the attribute
- NOT NULL: no NULL values allowed. NULL is default with no specification.

```
CREATE TABLE DEPARTMENT (
        DNAME                   VARCHAR(15)   UNIQUE,
        DNUMBER                 INT           PRIMARY KEY,
        MGRSSN                  CHAR(9)       NOT NULL,
        MGRSTARTDATE            DATE
);
CREATE TABLE DEPARTMENT (
        DNAME                   VARCHAR(15), …….
        UNIQUE (DNAME)
);
```

CREATE TABLE - CONSTRAINTS

- Attribute defaults: a default value for an attribute by appending DEFAULT <*value*>
  - The default value is included in any new tuple if an explicit value is not provided for that attribute.
  - If no default clause is specified, the default value is NULL for attributes that do not have the NOT NULL constraint.

```
CREATE TABLE WORKS_ON (
        ESSN            CHAR(9),
        PNO             INT,
        HOURS           DECIMAL(3,1) DEFAULT 10.0,
        PRIMARY KEY (ESSN, PNO)
);
```

CREATE TABLE - CONSTRAINTS

- Restrictions on attribute domains and other constraints on individual tuples within a table using the CHECK clause: *row-level constraints*

Constraint: Department numbers are restricted to integer numbers between 1 and 20. Starting date is from 2000-01-01.

```
CREATE TABLE DEPARTMENT (
        DNAME                   VARCHAR(15)   UNIQUE,
        DNUMBER                 INT           PRIMARY KEY
                        CHECK (DNUMBER>0 AND DNUMBER<21),
        MGRSSN                  CHAR(9)       NOT NULL,
        MGRSTARTDATE            DATE
                        CHECK (MGRSTARTDATE> '2000-01-01')
);
```

CREATE DOMAIN

- A domain can be declared, and the domain name can be used with the attribute specification. CREATE DOMAIN is *DBMS-specifically* implemented.
  - change the data type for a domain that is used by numerous attributes in a schema, and improve schema *readability*

CREATE **DOMAIN** *DomainName* AS *DataType*
[*CHECK* (<*condition*>)];

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);

CREATE DOMAIN D_NUM AS INTEGER
      CHECK (D_NUM > 0 AND D_NUM < 21);
```

CREATE DOMAIN

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);

CREATE DOMAIN D_NUM AS INTEGER
        CHECK (D_NUM > 0 AND D_NUM < 21);
```

```
CREATE TABLE DEPARTMENT (
        DNAME           VARCHAR(15)   UNIQUE,
        DNUMBER         D_NUM         PRIMARY KEY,
        MGRSSN          SSN_TYPE      NOT NULL,
        MGRSTARTDATE    DATE
);
```

What is difference between CHECK at the attribute level and CHECK at the domain level?

CREATE TABLE - CONSTRAINTS

- Giving *names* to constraints: CONSTRAINT
  - This is optional.
  - The name is unique within a particular database schema.
  - It is used to identify a particular constraint in case it must be dropped later and replaced with another one.
  - It is also possible to temporarily defer a constraint until the end of a transaction.

```
CREATE TABLE EMPLOYEE
        ( . . . ,
          DNO                 INT     NOT NULL     DEFAULT 1,
        CONSTRAINT EMPPK
          PRIMARY KEY (SSN) ,
        CONSTRAINT EMPSUPERFK
          FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)
                              ON DELETE SET NULL     ON UPDATE CASCADE ,
        CONSTRAINT EMPDEPTFK
          FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)
                              ON DELETE SET DEFAULT    ON UPDATE CASCADE );

CREATE TABLE DEPARTMENT
        ( . . . ,
          MGRSSN    CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
                  ON DELETE SET DEFAULT     ON UPDATE CASCADE );

CREATE TABLE DEPT_LOCATIONS
        ( . . . ,
          PRIMARY KEY (DNUMBER, DLOCATION),
          FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)
            ON DELETE CASCADE     ON UPDATE CASCADE ) ;
```

DROP Statement

- Used to drop *named* schema elements: schema (*database*), tables, indexes, functions, triggers,...
- Two drop behavior options:
  - CASCADE: all the elements referencing or contained inside the one being dropped are also dropped.
  - RESTRICT: only the one being dropped if no reference.
- DROP SCHEMA

  DROP SCHEMA *Company* CASCADE;

  DROP SCHEMA *Company* RESTRICT;

DROP Statement

- DROP TABLE:

  DROP TABLE *Dependent* CASCADE;

  - CASCADE: all such constraints, views, and other elements that reference the table are dropped automatically from the schema along with the table itself.

  DROP TABLE *Dependent* RESTRICT;

  - RESTRICT: dropped if it is not referenced in any constraints, views, and other elements.

DROP TABLE DEPARTMENT RESTRICT;

Error Code: 1217. Cannot delete or update a parent row: a foreign key constraint fails (MySQL)

ALTER Statement

- Modifications at the structure level:
  - ALTER DATABASE
  - ALTER TABLE
  - ALTER VIEW
  - ALTER FUNCTION
  - ALTER PROCEDURE
  - …
- ALTER TABLE:
  - COLUMN: ADD, MODIFY, DROP, …
  - CONSTRAINT: ADD, DROP, …
  - TABLE PROPERTIES
  - …

ALTER Statement

- *Add an attribute* for keeping track of jobs of employees to EMPLOYEE table in COMPANY schema

  ALTER TABLE Company.Employee ADD (Job VARCHAR(12));

  - Job value for each existing row: default value

  ALTER TABLE Company.Employee ADD (Job VARCHAR(12) DEFAULT 'EMPLOYEE');

  - What value does each existing row take for attribute Job if?

  ALTER TABLE Company.Employee ADD (Job VARCHAR(12) NOT NULL);

  - What happens to *two or more existing rows* if attribute Job is *not null and unique*?

  ALTER TABLE Company.Employee ADD (Job VARCHAR(12) *NOT NULL UNIQUE*);

ALTER Statement

- Drop a column: similarly to drop a table, CASCADE or RESTRICT option must be specified
  - CASCADE option: all constraints and views referencing the column are dropped along with the column
  - RESTRICT option: successful only if no constraints and views are referencing the column

  ALTER TABLE Company.Employee DROP [COLUMN] Address CASCADE;

- What if a *primary key* is dropped?

  ALTER TABLE Company.Employee DROP Ssn CASCADE;

ALTER Statement

- Add a constraint to a table. Added constraints must be satisfied by the current content.
  - Primary key

```
ALTER TABLE Works_on ADD PRIMARY KEY (Ssn, Pnumber);
```

  - Foreign key

```
ALTER TABLE Works_on ADD CONSTRAINT ssn_fk
FOREIGN KEY (Ssn) REFERENCES Employee (Ssn)
ON DELETE CASCADE ON UPDATE CASCADE;
```

  - Default values

```
ALTER TABLE Employee ALTER COLUMN Dno SET DEFAULT 1;
```

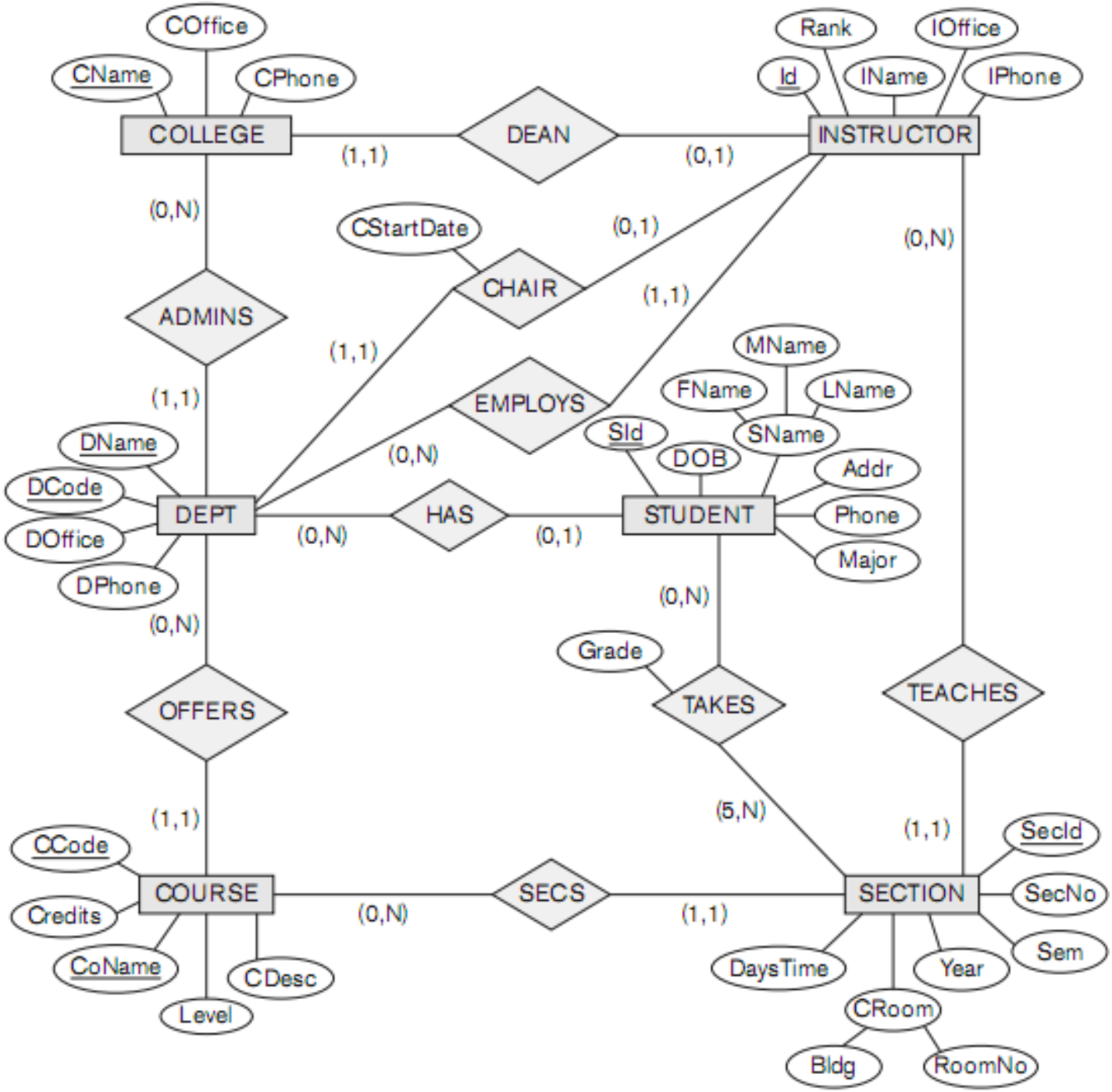
  - Unique

```
ALTER TABLE Department ADD UNIQUE (Dname);
```

Data Definition Language

- Drop a constraint

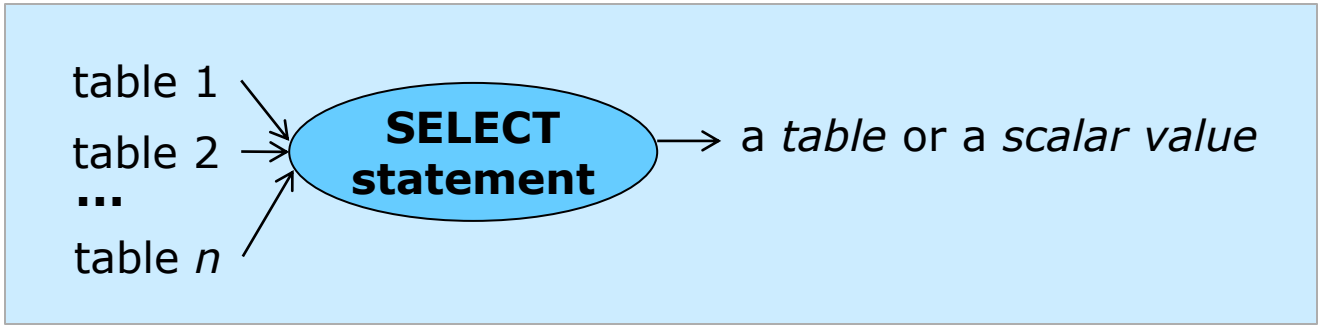
```
ALTER TABLE Works_on
DROP PRIMARY KEY;

ALTER TABLE Works_on
DROP FOREIGN KEY ssn_fk CASCADE;

ALTER TABLE Employee
ALTER COLUMN Dno DROP DEFAULT;

//Oracle
ALTER TABLE Department
DROP UNIQUE (Dname);

//MySQL – DROP UNIQUE
ALTER TABLE Department
DROP INDEX Dname;
```



Write DDL statements for the UNIVERSITY database on a *specific* DBMS. Pay attention to the *references* among the tables and their *other constraints*.

SELECT Statement

- SQL has one basic statement for retrieving information from a database: SELECT statement.
- SELECT statement is *not the same as* SELECTION operation (σ) of the relational algebra.
- Important distinction between the SQL data model and the formal relational model is that: SQL allows a table to have two or more tuples that are identical in all their attribute values.
  - A SQL table is a *multi-set* (sometimes called a bag) of tuples, which *is not* a set of tuples.
  - A SQL table is a set of tuples, like a relation in the relational model, once a primary key is defined on it.



SELECT Statement

SELECT [DISTINCT | ALL]

{* | [columnExpression [AS newName]] | built-inFunction [,...] }

[**FROM** TableName|*ViewName* [alias] [, ...]]

[**WHERE** rowCondition]

[**GROUP BY** columnList]

[**HAVING** groupCondition]

[**ORDER BY** columnList | columnPositionList [ASC|DESC]]

SELECT Statement

Typical processing of a query block

Execution order	Clause	Meaning
1	FROM	Specifies *and joins* table(s)/ view(s) to be used
2	WHERE	Filters rows by row-level conditions
3	GROUP BY	Forms groups of rows with the same grouping column value
4	HAVING	Filters groups by group-level conditions
5	SELECT	Specifies the output to be returned
6	ORDER BY	Specifies the order of the output

The COMPANY database

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Source: [1]

SELECT Statement

For each project in Stafford, *on which more than two employees work*, retrieve the project name and the number of employees who work on that project in descending order of the project name.

```
SELECT      PNAME, COUNT (*)
FROM        PROJECT JOIN WORKS_ON ON PNUMBER=PNO
WHERE       PLOCATION = 'Stafford'
GROUP BY    PNAME
HAVING      COUNT (*) > 2
ORDER BY    PNAME DESC;
```

SELECT Statement

FROM PROJECT **JOIN** WORKS_ON **ON** PNUMBER=PNO

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

⋈ PNUMBER=PNO

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

SELECT Statement

FROM PROJECT **JOIN** WORKS_ON **ON** PNUMBER=PNO

Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
ProductX	1	Bellaire	5	123456789	1	32.5
ProductX	1	Bellaire	5	453453453	1	20.0
ProductY	2	Sugarland	5	123456789	2	7.5
ProductY	2	Sugarland	5	453453453	2	20.0
ProductY	2	Sugarland	5	333445555	2	10.0
ProductZ	3	Houston	5	666884444	3	40.0
ProductZ	3	Houston	5	333445555	3	10.0
Computerization	10	Stafford	4	333445555	10	10.0
Computerization	10	Stafford	4	999887777	10	10.0
Computerization	10	Stafford	4	987987987	10	35.0
Reorganization	20	Houston	1	333445555	20	10.0
Reorganization	20	Houston	1	987654321	20	15.0
Reorganization	20	Houston	1	888665555	20	NULL
Newbenefits	30	Stafford	4	999887777	30	30.0
Newbenefits	30	Stafford	4	987987987	30	5.0
Newbenefits	30	Stafford	4	987654321	30	20.0

SELECT Statement

WHERE PLOCATION = 'Stafford'

Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
ProductX	1	Bellaire	5	123456789	1	32.5
ProductX	1	Bellaire	5	453453453	1	20.0
ProductY	2	Sugarland	5	123456789	2	7.5
ProductY	2	Sugarland	5	453453453	2	20.0
ProductY	2	Sugarland	5	333445555	2	10.0
ProductZ	3	Houston	5	666884444	3	40.0
ProductZ	3	Houston	5	333445555	3	10.0
Computerization	10	Stafford	4	333445555	10	10.0
Computerization	10	Stafford	4	999887777	10	10.0
Computerization	10	Stafford	4	987987987	10	35.0
Reorganization	20	Houston	1	333445555	20	10.0
Reorganization	20	Houston	1	987654321	20	15.0
Reorganization	20	Houston	1	888665555	20	NULL
Newbenefits	30	Stafford	4	999887777	30	30.0
Newbenefits	30	Stafford	4	987987987	30	5.0
Newbenefits	30	Stafford	4	987654321	30	20.0

SELECT Statement

WHERE PLOCATION = 'Stafford'

Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
Computerization	10	Stafford	4	333445555	10	10.0
Computerization	10	Stafford	4	999887777	10	10.0
Computerization	10	Stafford	4	987987987	10	35.0
Newbenefits	30	Stafford	4	999887777	30	30.0
Newbenefits	30	Stafford	4	987987987	30	5.0
Newbenefits	30	Stafford	4	987654321	30	20.0

SELECT Statement

GROUP BY PNAME

	Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
Group 1	Computerization	10	Stafford	4	333445555	10	10.0
	Computerization	10	Stafford	4	999887777	10	10.0
	Computerization	10	Stafford	4	987987987	10	35.0
Group 2	Newbenefits	30	Stafford	4	999887777	30	30.0
	Newbenefits	30	Stafford	4	987987987	30	5.0
	Newbenefits	30	Stafford	4	987654321	30	20.0

SELECT Statement

HAVING COUNT (*) > 2

	Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
Group 1	Computerization	10	Stafford	4	333445555	10	10.0
	Computerization	10	Stafford	4	999887777	10	10.0
	Computerization	10	Stafford	4	987987987	10	35.0
Group 2	Newbenefits	30	Stafford	4	999887777	30	30.0
	Newbenefits	30	Stafford	4	987987987	30	5.0
	Newbenefits	30	Stafford	4	987654321	30	20.0

Group 1: COUNT(*) = 3 > 2 => TRUE => group 1 selected

Group 2: COUNT(*) = 3 > 2 => TRUE => group 2 selected

SELECT Statement

SELECT PNAME, COUNT (*)

	Pname	Pnumber	Plocation	Dnum	Essn	Pno	Hours
Group 1	Computerization	10	Stafford	4	333445555	10	10.0
	Computerization	10	Stafford	4	999887777	10	10.0
	Computerization	10	Stafford	4	987987987	10	35.0
	Newbenefits	30	Stafford	4	999887777	30	30.0
Group 2	Newbenefits	30	Stafford	4	987987987	30	5.0
	Newbenefits	30	Stafford	4	987654321	30	20.0

Group 1: COUNT(*) = 3 > 2 => TRUE => group 1 selected

Group 2: COUNT(*) = 3 > 2 => TRUE => group 2 selected

Pname	COUNT(*)
Computerization	3
Newbenefits	3

SELECT Statement

ORDER BY PNAME **DESC**;

Pname	COUNT(*)
Computerization	3
Newbenefits	3

Descending order

Pname	COUNT(*)
Newbenefits	3
Computerization	3

For each project in Stafford, *on which more than two employees work*, retrieve the project name and the number of employees who work on that project in descending order of the project name.

```
SELECT      PNAME, COUNT (*)
FROM        PROJECT JOIN WORKS_ON ON PNUMBER=PNO
WHERE       PLOCATION = 'Stafford'
GROUP BY    PNAME
HAVING      COUNT (*) > 2
ORDER BY    PNAME DESC;
```

Input Tables

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

Query Result

Pname	COUNT(*)
Newbenefits	3
Computerization	3

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Retrieve all the information of each employee who is with department 4 and salary > 25000 or with department 5 and salary > 30000.

$\sigma_{\textbf{(Dno=4 AND Salary>25000) OR (Dno=5 AND Salary>30000)}}\textbf{(EMPLOYEE)}$

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

```
SELECT *
FROM EMPLOYEE
WHERE (Dno = 4 AND Salary > 25000) OR (Dno = 5 AND SALARY > 30000);
```

* (*asterisk*): all columns in the previous process are included in the output.

The *primary key in the output* => the output is a *set* (*relation*).

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, T				
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, T				

Retrieve Ssn, date of birth, and department number of each employee.

$\pi_{\text{Ssn, Bdate, Dno}}$ **(EMPLOYEE)**

Ssn	Bdate	Dno
123456789	1965-01-09	5
333445555	1955-12-08	5
999887777	1968-01-19	4
987654321	1941-06-20	4
666884444	1962-09-15	5
453453453	1972-07-31	5
987987987	1969-03-29	4
888665555	1937-11-10	1

```
SELECT Ssn, Bdate, Dno
FROM EMPLOYEE;
```

List all the department numbers of the employees.

π_{Dno} **(EMPLOYEE)**

Dno
5
4
1

```
SELECT DISTINCT Dno
FROM EMPLOYEE;
```

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Return all the combinations of departments and their locations.
DEPARTMENT x DEPT_LOCATIONS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Dnumber	Dlocation
Research	5	333445555	1988-05-22	1	Houston
Research	5	333445555	1988-05-22	4	Stafford
Re					Bellaire
Re					Sugarland
Re					Houston
Admi					Houston
Admi					Stafford
Admi					Bellaire
Admi					Sugarland
Admi					Houston
Headquarters	1	888665555	1981-06-19	1	Houston
Headquarters	1	888665555	1981-06-19	4	Stafford
Headquarters	1	888665555	1981-06-19	5	Bellaire
Headquarters	1	888665555	1981-06-19	5	Sugarland
Headquarters	1	888665555	1981-06-19	5	Houston

```
SELECT *
FROM DEPARTMENT, DEPT_LOCATIONS;

SELECT *
FROM DEPARTMENT CROSS JOIN DEPT_LOCATIONS;
```

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Return all the combinations of Research department with Houston location.

DEPARTMENT $\bowtie_{\text{Dname = 'Research' AND Dlocation = 'Houston'}}$ **DEPT_LOCATIONS**

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Dnumber	Dlocation
Research	5	333445555	1988-05-22	1	Houston
Research	5	333445555	1988-05-22	5	Houston

```
SELECT *
FROM DEPARTMENT, DEPT_LOCATIONS;
WHERE Dname = 'Research' AND Dlocation = 'Houston';

SELECT *
FROM DEPARTMENT CROSS JOIN DEPT_LOCATIONS;
WHERE Dname = 'Research' AND Dlocation = 'Houston';
```

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Retrieve all the information of each department and its locations.
DEPARTMENT $\bowtie_{\text{Dnumber = Dnumber}}$ **DEPT_LOCATIONS**

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Dnumber	Dlocation
Research	5	333445555	1988-05-22	5	Bellaire
Research	5	333445555	1988-05-22	5	Sugarland
Research	5	333445555	1988-05-22	5	Houston
Administration	4	987654321	1995-01-01	4	Stafford
Headquarters	1	888665555	1981-06-19	1	Houston

```
SELECT *
FROM DEPARTMENT d, DEPT_LOCATIONS dl;
WHERE d.Dnumber = dl.Dnumber;

SELECT *
FROM DEPARTMENT d CROSS JOIN DEPT_LOCATIONS dl;
WHERE d.Dnumber = dl.Dnumber;

SELECT *
FROM DEPARTMENT d JOIN DEPT_LOCATIONS dl ON d.Dnumber = dl.Dnumber;
```

Ambiguous names => qualified names by [schemaName.]tableName.attributeName
ALIAS => alternative table name => tuple variable for the table

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Retrieve all the information of each department and its locations.
DEPARTMENT * DEPT_LOCATIONS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Dlocation
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston
Administration	4	987654321	1995-01-01	Stafford
Headquarters	1	888665555	1981-06-19	Houston

```
SELECT *
FROM DEPARTMENT NATURAL JOIN DEPT_LOCATIONS;
```

Return the information of all the employees and their departments they manage.

EMPLOYEE $\bowtie_{\text{Ssn = Mgr\_ssn}}$ **DEPARTMENT**

Fname	Minit	Lname	Ssn	B date	Address	Sex	Salary	Super_ ssn	Dno	Dname	Dnumber	Mgr_ ssn	Mgr_ start_ date
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5	Research	5	333445555	1988-05-22
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4	Administration	4	987654321	1995-01-01
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1	Headquaters	1	888665555	1981-06-09

```
SELECT *
FROM EMPLOYEE, DEPARTMENT;
WHERE Ssn = Mgr_ssn;

SELECT *
FROM EMPLOYEE CROSS JOIN DEPARTMENT;
WHERE Ssn = Mgr_ssn;

SELECT *
FROM EMPLOYEE JOIN DEPARTMENT ON Ssn = Mgr_ssn;
```

Return the information of all the employees and their departments that they manage if any.

EMPLOYEE ⟕ **Ssn = Mgr_ssn** **DEPARTMENT**

Fname	Minit	Lname	Ssn	B date	Address	Sex	Salary	Super_ ssn	Dno	Dname	Dnumber	Mgr_ ssn	Mgr_ start_ date
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5	Research	5	333445555	1988-05-22
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4	Administration	4	987654321	1995-01-01
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1	Headquaters	1	888665555	1981-06-09
John	B	Smith	123456789	1965-01-09	732 Fondren, Houston, TX	M	30000	333445555	5	NULL	NULL	NULL	NULL
Alicia	J	Zelaya	999977888	…	…	…	…	…	4	NULL	NULL	NULL	NULL
Ramesh	K	Narayan	…	…	…	…	…	…	5	NULL	NULL	NULL	NULL
Joyce	A	English	…	…	…	…	…	…	5	NULL	NULL	NULL	NULL
Ahmad	V	Jabbar	…	…	…	…	…	…	4	NULL	NULL	NULL	NULL

```
SELECT *
FROM EMPLOYEE LEFT OUTER JOIN DEPARTMENT ON Ssn = Mgr_ssn;
```

SEMI-JOIN: $T1 \ltimes_{T1.X=T2.Y} T2$

Return the information of all the departments *with at least one* employee who has salary > 30000.

DEPARTMENT $\ltimes_{\text{Dnumber=Dno}}$ $(\sigma_{\text{Salary>30000}}(\text{EMPLOYEE}))$

```
SELECT Dname, Dnumber, Mgr_ssn, Mgr_start_date
FROM DEPARTMENT JOIN (SELECT * FROM EMPLOYEE WHERE Salary>30000) e
ON Dnumber = Dno;
```

DNAME	DNUMBER	MGR_SSN	MGR_START_DATE
Headquarters	1	888665555	1981-06-19
Administration	4	987654321	1995-01-01
Research	5	333445555	1988-05-22

```
SELECT *
FROM DEPARTMENT
WHERE Dnumber IN (SELECT Dno FROM EMPLOYEE WHERE Salary > 30000);
```

```
SELECT *
FROM DEPARTMENT d
WHERE EXISTS
   (SELECT * FROM EMPLOYEE WHERE Dno = d.Dnumber AND Salary > 30000);
```

SEMI-JOIN
(Oracle, MySQL, …)
(*DBMS-specific implementation*)

EXISTS vs. NOT EXISTS

- EXISTS function returns TRUE or FALSE.
  - TRUE: if the nested query result contains *at least one* tuple
  - FALSE: if the nested query result contains *no* tuples

Retrieve Ssn and name of each employee who has *no* dependents.

```
SELECT Ssn, Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
                  FROM DEPENDENT
                  WHERE Essn = Ssn);

SELECT Ssn, Fname, Lname
FROM EMPLOYEE
WHERE Ssn NOT IN (SELECT DISTINCT ESsn
                  FROM DEPENDENT);
```

Ssn	Fname	Lname
453453453	Joyce	English
666884444	Ramesh	Narayan
888665555	James	Borg
987987987	Ahmad	Jabbar
999887777	Alicia	Zelaya

EXISTS vs. NOT EXISTS

- EXISTS function returns TRUE or FALSE.
  - TRUE: if the nested query result contains *at least one* tuple
  - FALSE: if the nested query result contains *no* tuples

List Ssn and name of each manager who has ***at least one*** dependent.

```
SELECT Ssn, Fname, Lname
FROM EMPLOYEE
WHERE EXISTS (SELECT *
                FROM DEPENDENT
                WHERE Essn = Ssn)
   AND EXISTS (SELECT *
                FROM DEPARTMENT
                WHERE Mgr_ssn = Ssn);
```

Ssn	Fname	Lname
333445555	Franklin	Wong
987654321	Jennifer	Wallace

ANTI JOIN: T1 $\overline{\ltimes}_{X = Y}$ T2

Return the information of all the employees who *do not* work in any department which was managed since 1990.

EMPLOYEE $\overline{\ltimes}_{\text{Dno=Dnumber}}$ **(**$\sigma_{\text{Mgr\_start\_date>='1990-01-01'}}$**(DEPARTMENT))**

```
SELECT *
FROM EMPLOYEE
WHERE Dno NOT IN ( SELECT Dnumber
                   FROM DEPARTMENT
                   WHERE Mgr_start_date >= '1990-01-01');
```

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1

IN vs. NOT IN

- IN: comparison operator, which compares a value *v* with a set (or multiset) of values *V* and evaluates to TRUE if *v* is one of the elements in *V*.
  - *V*: a nested query or an explicit set
  - = ANY (or = SOME) operator: equivalent to IN operator
- NOT IN: TRUE if *v* is not any element in *V*.

List Essns of all employees who work the same (project, hours) combination on some project that employee whose Ssn is '123456789' works on

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN (SELECT Pno, Hours
                       FROM WORKS_ON
                       WHERE Essn = '123456789');
```

Essn
123456789

IN vs. NOT IN

List the project numbers and names of projects that have an employee with last name 'Smith' involved as worker or as manager of the department controlling the projects.

Pnumber	Pname
1	ProductX
2	ProductY

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

IN vs. NOT IN

- It is also possible to use an *explicit* (*enumerated*) *set of values* with IN comparison operator.

Retrieve the SSNs of all employees who work on project numbers 1, 2, or 3.

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN  (1, 2, 3);

SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO = 1 OR PNO = 2 OR PNO = 3;
```

ESSN
123456789
453453453
333445555
666884444

UNION: R U S

- *Union compatibility*: R and S has the same degree and each corresponding pair of attributes has the same domain.
- Produce a new relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

RESULT1 U RESULT2

RESULT1	SSN
	123456789
	333445555
	666884444
	453453453

RESULT2	SSN
	333445555
	888665555

RESULT	SSN
	123456789
	333445555
	666884444
	453453453
	888665555

```
SELECT * FROM RESULT1
UNION
SELECT * FROM RESULT2;
```

Check RESULT for ***UNION ALL***?

UNION: R U S

Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
(SELECT    DISTINCT PNUMBER
FROM       PROJECT, DEPARTMENT, EMPLOYEE
WHERE      DNUM=DNUMBER AND MGR_SSN=SSN
           AND LNAME='Smith')
UNION
(SELECT    DISTINCT PNUMBER
FROM       PROJECT, WORKS_ON, EMPLOYEE
WHERE      PNUMBER=PNO AND ESSN=SSN
           AND LNAME='Smith');
```

PNUMBER
1
2

USE IN OPERATOR ALTERNATIVELY

(*DBMS-specific implementation*)

INTERSECTION: R ∩ S

- *Union compatibility*: R and S has the same degree and each corresponding pair of attributes has the same domain.
- Produce a new relation that includes all tuples that are in both R and S.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

STUDENT ∩ INSTRUCTOR

FN	LN
Susan	Yao
Ramesh	Shah

```
SELECT * FROM STUDENT
INTERSECT
SELECT * FROM INSTRUCTOR;
```

Check RESULT for ***INTERSECT ALL***?

(*DBMS-specific implementation*)

DIFFERENCE: R – S

USE NOT IN OPERATOR ALTERNATIVELY (*DBMS-specific implementation*)

- *Union compatibility*: R and S has the same degree and each corresponding pair of attributes has the same domain.
- Produce a new relation that includes all tuples that are in R but not in S.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

STUDENT - INSTRUCTOR

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR - STUDENT

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

```
SELECT * FROM STUDENT
MINUS
SELECT * FROM INSTRUCTOR;
```

```
SELECT * FROM INSTRUCTOR
MINUS
SELECT * FROM STUDENT;
```

DIVISION: T(Y) = R(Z) ÷ S(X), where X ⊆ Z

- Produce a new relation T each tuple of which appears in R in combination with *every* tuple in S => a sequence of π, x, –

The projects of all employees

SSN_PNOS	ESSN	PNO
	123456789	1
	123456789	2
	666884444	3
	453453453	1
	453453453	2
	333445555	2
	333445555	3
	333445555	10
	333445555	20
	999887777	30
	999887777	10
	987987987	10
	987987987	30
	987654321	30
	987654321	20
	888665555	20

Smith's projects

SMITH_PNOS	PNO
	1
	2

Return the employees who worked on *all* the projects that Smith worked:

SSN_PNOS ÷ SMITH_PNOS

$T1 = \pi_Y(R)$

$T2 = \pi_Y(T1 \times S - R)$

$T = T1 - T2$

SSNS	SSN
	123456789
	453453453

```
CREATE TABLE T1 AS
SELECT DISTINCT ESSN
FROM SSN_PNOS;

CREATE TABLE T2 AS
SELECT DISTINCT ESSN
FROM ( SELECT * FROM T1, SMITH_PNOS
       MINUS
       SELECT * FROM SSN_PNOS);

SELECT * FROM T1
MINUS
SELECT * FROM T2;
```

AGGREGATION: <a grouping attribute list> ℑ <function list> (R)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

How many employees work in the company?

ℑ COUNT Ssn (EMPLOYEE)

How many employees work in each department of the company?

Dno ℑ COUNT Ssn (EMPLOYEE)

```
SELECT COUNT(Ssn) AS COUNT_Ssn
FROM EMPLOYEE;
```

COUNT_Ssn
8

```
SELECT Dno, COUNT(Ssn) AS COUNT_Ssn
FROM EMPLOYEE
GROUP BY Dno;
```

Dno	COUNT_Ssn
5	4
4	3
1	1

SELECT Statement – Substring Pattern Matching

- *Substring pattern matching*: allows comparison conditions on only parts of a character string, using the LIKE comparison operator.
- Partial strings are specified by using any of two reserved characters: **%** and _ (*underscore*)
  - **%**: replaces an arbitrary number of zero or more characters
  - _ : replaces a single character

```
SELECT Ssn, Fname, Lname, Address
FROM EMPLOYEE
WHERE Address LIKE '%Houston, TX%';
```

Ssn	Fname	Lname	Address
123456789	John	Smith	731 Fondren, Houston, TX
333445555	Franklin	Wong	638 Voss, Houston, TX
453453453	Joyce	English	5631 Rice, Houston, TX
888665555	James	Borg	450 Stone, Houston, TX
987987987	Ahmad	Jabbar	980 Dallas, Houston, TX

```
SELECT Ssn, Fname, Lname, BDate
FROM Employee
WHERE BDate LIKE '_ _ _ _ _ _ 0 1 _ _ _ _';
```

Ssn	Fname	Lname	BDate
123456789	John	Smith	1965-01-09
999887777	Alicia	Zelaya	1968-01-19

SELECT Statement – Arithmetic Operators

- Standard arithmetic operators: +, -, *, /

Show the resulting salaries if every employee working on "ProductX" project is given 10% raise.

```
SELECT FNAME, LNAME, 1.1*Salary AS INC_SAL
FROM Employee, Works_on, Project
WHERE SSN=ESSN
    AND PNO=PNUMBER
    AND PNAME='ProductX';
```

FNAME	LNAME	INC_SAL
John	Smith	33000.000
Joyce	English	27500.000

SELECT Statement - NULL

- Comparison operators for NULL values
  - IS NULL: checks whether an attribute value is NULL
  - IS NOT NULL: checks whether an attribute value is NOT NULL

Retrieve Ssn, names, Super_ssn of all employees who *do not have* supervisors.

SELECT Ssn, Fname, Lname, Super_SSN

FROM EMPLOYEE

WHERE Super_SSN **IS NULL**;

Ssn	Fname	Lname	Super_SSN
888665555	James	Borg	NULL

Retrieve Ssn, Fname, Lname, Super_SSN of all employees who *have* supervisors.

SELECT Ssn, Fname, Lname, Super_SSN

FROM EMPLOYEE

WHERE Super_SSN **IS NOT NULL**;

Ssn	Fname	Lname	Super_SSN
123456789	John	Smith	333445555
333445555	Franklin	Wong	888665555
453453453	Joyce	English	333445555
666884444	Ramesh	Narayan	333445555
987654321	Jennifer	Wallace	888665555
987987987	Ahmad	Jabbar	987654321
999887777	Alicia	Zelaya	987654321

SELECT Statement – More Comparison Operators

- =, >, >=, <, <=, <>, BETWEEN
  - *X* BETWEEN *v*1 AND *v*2: *X* >= *v*1 AND *X* <= *v*2
- ANY, SOME, ALL
  - The comparison condition (*v* > ALL *V*) returns TRUE if the value v is greater than all the values in the set (or multiset) *V*.

Retrieve Ssn Fname, Lname, Dnumber, and Salary of each employee in department 5 whose salary is between $30,000 and $40,000.

SELECT Ssn, Fname, Lname, Dno Dnumber, Salary

FROM EMPLOYEE

WHERE Salary **BETWEEN** 30000 **AND** 40000 **AND** Dno=5;

Ssn	Fname	Lname	Dnumber	Salary
123456789	John	Smith	5	30000.00
333445555	Franklin	Wong	5	40000.00
666884444	Ramesh	Narayan	5	38000.00

SELECT Statement – More Comparison Operators

- =, >, >=, <, <=, <>, BETWEEN
- ANY, SOME, ALL
  - The comparison condition (*v* > ALL *V*) returns TRUE if the value *v* is greater than all the values in the set (or multiset) *V*.

Retrieve Ssn, Fname, Lname, and Salary of each employee whose salary is greater than the salary of all employees in department 5.

SELECT Ssn, Fname, Lname, Salary

FROM EMPLOYEE

WHERE Salary **> ALL** (**SELECT** Salary **FROM** Employee **WHERE** Dno=5);

Dno	MAX_Salary
5	40000.00

SELECT Ssn, Fname, Lname, Salary

FROM EMPLOYEE

WHERE Salary **>**(**SELECT** MAX(Salary) **FROM** Employee **WHERE** Dno=5);

Ssn	Fname	Lname	Salary
888665555	James	Borg	55000.00
987654321	Jennifer	Wallace	43000.00

SELECT Statement – Multiway Join

- It is also possible to *nest join specifications*; that is, one of the tables in a join may itself be a joined table.
- Nesting allows the specification of the join of three or more tables as a single joined table, called a *multiway join*.

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

```
SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGR_SSN=SSN AND PLOCATION='Stafford';

SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS
FROM   (PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
                          JOIN EMPLOYEE ON MGR_SSN=SSN
WHERE PLOCATION='Stafford';
```

PNUMBER	DNUM	LNAME	BDATE	ADDRESS
10	4	Wallace	1941-06-20	291 Berry, Bellaire, TX
30	4	Wallace	1941-06-20	291 Berry, Bellaire, TX

SELECT Statement – Nested Queries

- Nested queries: complete [select-from-where-…] query blocks *within* another query which is called the *outer query*.
- These nested queries can also appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clauses as needed.
- If a nested query returns a single attribute and a single tuple, the result will be a single (*scalar*) value. In such cases, it is permissible to use = instead of IN for the comparison operator.
- In general, the nested query will return a *table*, which is a set or multiset of tuples.
- SQL allows the use of tuples of values in comparisons by placing them within *parentheses*.

List the name of the department where Smith works.

Dname
Research

SELECT Dname
FROM DEPARTMENT
WHERE Dnumber **IN** (**SELECT** Dno **FROM** EMPLOYEE **WHERE** Lname = 'Smith');

SELECT Statement – Nested Queries

Retrieve Ssn, name, and address of all employees whose salary is greater than the average salary of the employees in 'Research' department.

```
SELECT SSN, FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE SALARY > (SELECT AVG(SALARY)
                FROM EMPLOYEE
                WHERE DNO IN (SELECT DNUMBER
                              FROM DEPARTMENT
                              WHERE DNAME='Research'));
```

SSN	FNAME	LNAME	ADDRESS
333445555	Franklin	Wong	638 Voss, Houston, TX
666884444	Ramesh	Narayan	975 Fire Oak, Humble, TX
888665555	James	Borg	450 Stone, Houston, TX
987654321	Jennifer	Wallace	291 Berry, Bellaire, TX

For each department that has more than two employees, retrieve the department number and the number of its employees who are making more than $30,000.

```
SELECT Dno, count(*) ENumber
FROM EMPLOYEE
WHERE Salary>30000 AND Dno IN (SELECT Dno
                               FROM EMPLOYEE
                               GROUP BY Dno
                               HAVING count(*) > 2)
GROUP BY Dno;
```

Dno	ENumber
4	1
5	2

SELECT Statement – Correlated Nested Queries

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*.
- A *correlated query* is the query where the nested query is evaluated *once for each tuple (or combination of tuples) in the outer query*.

List Ssn, name, and department number of each employee who has the salary greater than the average salary of the employees in his/her department.

```
SELECT Ssn, Fname, Lname, Dno Dnumber
FROM EMPLOYEE E
WHERE Salary > (SELECT AVG(Salary)
                FROM EMPLOYEE
                WHERE Dno = E.Dno);
```

Ssn	Fname	Lname	Dnumber
333445555	Franklin	Wong	5
666884444	Ramesh	Narayan	5
987654321	Jennifer	Wallace	4

SELECT Statement – Correlated Nested Queries

Retrieve the name of each employee who has a dependent with the same first name as the employee.

FNAME	LNAME

```
SELECT FNAME, LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
                FROM DEPENDENT
                WHERE ESSN=E.SSN
                    AND DEPENDENT_NAME=E.FNAME);
```

Retrieve Ssn and name of each employee who has two or more dependents.

Ssn	Lname	Fname
123456789	Smith	John
333445555	Wong	Franklin

```
SELECT Ssn, Lname, Fname
FROM EMPLOYEE
WHERE (SELECT count(*)
       FROM DEPENDENT
       WHERE Essn = Ssn) >= 2;
```

INSERT Statement

- In its simplest form, INSERT is used to add one row in a table.
- Table name and a list of values for a new row need to be specified.
- Attribute values must be listed *in the same order* as the attributes were specified in the CREATE TABLE statement.
- NOTE: All the DDL *constraints* on the table are checked for any modification on the table.
  - A *DBMS* that fully implements SQL should *support and enforce all the integrity constraints* that can be specified in the *DDL*.

INSERT Statement

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

```
INSERT INTO EMPLOYEE
VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30',
'98  Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);
```

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Richard	K	Marini	653298653	1962-12-30	98 Oak Forest, Katy, TX	M	37000.00	653298653	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

INSERT Statement

- A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.
  - The values must include all attributes with NOT NULL specification and no default value.
  - Attributes with NULL allowed or DEFAULT values are the ones that can be left out.

INSERT Statement

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Richard	K	Marini	653298653	1962-12-30	98 Oak Forest, Katy, TX	M	37000.00	653298653	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno, Ssn)
VALUES ('Peter', 'Pan', 1, '123123123');
```

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
Peter	NULL	Pan	123123123	NULL	NULL	NULL	NULL	NULL	1
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Richard	K	Marini	653298653	1962-12-30	98 Oak Forest, Katy, TX	M	37000.00	653298653	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

INSERT Statement

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno)
VALUES ('Jack', 'Sparrow', 1);
```

Error Code: 1364. Field 'SSN' doesn't have a default value

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno, Ssn)
VALUES ('Jack', 'Sparrow', 1, '123456789');
```

Error Code: 1062. Duplicate entry '123456789' for key 'PRIMARY'

```
INSERT INTO EMPLOYEE
VALUES ('James', 'J', 'Watson', '653298653', '1962-12-30',
'98  Oak Forest, Houston, TX', 'M', 37000, '653298653', NULL);
```

Error Code: 1048. Column 'DNO' cannot be null

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno, Ssn)
VALUES ('Jack', 'Sparrow', 2, '234234234');
```

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`company`.`employee`, CONSTRAINT `employee_ibfk_2` FOREIGN KEY (`DNO`) REFERENCES `department` (`DNUMBER`))

INSERT Statement

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno)
VALUES ('Jack', 'Sparrow', 1);
```

Error Code: 1364. Field 'SSN' doesn't have a default value

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno, Ssn)
VALUES ('Jack', 'Sparrow', 1, '123456789');
```

Error Code: 1062. Duplicate entry '123456789' for key 'PRIMARY'

```
INSERT INTO EMPLOYEE
VALUES ('James', 'J', 'Watson', '653298653', '1962-12-30',
'98  Oak Forest, Houston, TX', 'M', 37000, '653298653', NULL);
```

Error Code: 1048. Column 'DNO' cannot be null

```
INSERT INTO EMPLOYEE  (Fname, Lname, Dno, Ssn)
VALUES ('Jack', 'Sparrow', 2, '234234234');
```

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails

HOW TO FIX THEM?

INSERT Statement

- A variation of the INSERT command *inserts multiple tuples into a table* in conjunction with creating the table and loading it with the result of a query.

```
CREATE TABLE WORKS_ON_INFO (
        Emp_name            VARCHAR(15),
        Proj_name           VARCHAR(15),
        Hours_per_week      DECIMAL(3,1)
);

INSERT INTO WORKS_ON_INFO
SELECT  E.Lname, P.Pname, W.Hours
FROM  PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE  P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

INSERT Statement

- Another variation for loading data is to create a new table TNEW that has *the same attributes* as an existing table T, and load some of the data currently in T into TNEW. The syntax for doing this uses the *LIKE* clause.

```
CREATE TABLE D5EMPS LIKE EMPLOYEE;

INSERT INTO D5EMPS
SELECT  *
FROM  EMPLOYEE
WHERE  Dno = 5;
```

? COMPARED with:

```
CREATE TABLE D5EMPS AS
SELECT  *
FROM  EMPLOYEE
WHERE  Dno = 5;
```

DELETE Statement

- The DELETE command removes tuples from a relation.
- It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time. However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL.
- A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.
  - The DROP TABLE command is used to remove the table definition.

DELETE Statement

Remove from EMPLOYEE table the details of the employees whose last name is Brown.

```
DELETE FROM EMPLOYEE
WHERE       LNAME='Brown';
```

How many rows are deleted?
Is any constraint violated?

Remove from EMPLOYEE table the details of the employee whose SSN is 123456789.

```
DELETE FROM EMPLOYEE
WHERE       SSN='123456789';
```

Remove from EMPLOYEE table the details of all the employees who work in Research department.

```
DELETE FROM EMPLOYEE
WHERE       DNO IN (SELECT DNUMBER
                    FROM DEPARTMENT
                    WHERE DNAME = 'Research');
```

Remove from EMPLOYEE table the details of all the employees.

```
DELETE FROM EMPLOYEE;
```

UPDATE Statement

- The UPDATE command is used to *modify attribute values of one or more selected tuples in a table*.
- As in the DELETE command, a *WHERE clause* in the UPDATE command selects the tuples to be modified from a single table.
  - Updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL.
  - To modify multiple relations, several UPDATE commands must be issued.
- The *SET clause* in the UPDATE command specifies the attributes to be modified and their new values.

UPDATE Statement

Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
UPDATE      PROJECT
SET         PLOCATION = 'Bellaire', DNUM = 5
WHERE       PNUMBER=10;
```

Increase 2 hours for all the employees of department 1 working on the projects.

```
UPDATE      WORKS_ON
SET         HOURS = HOURS + 2
WHERE       ESSN IN (SELECT DISTINCT SSN
                     FROM EMPLOYEE
                     WHERE DNO = 1);
```

Data Control Language - Access Control

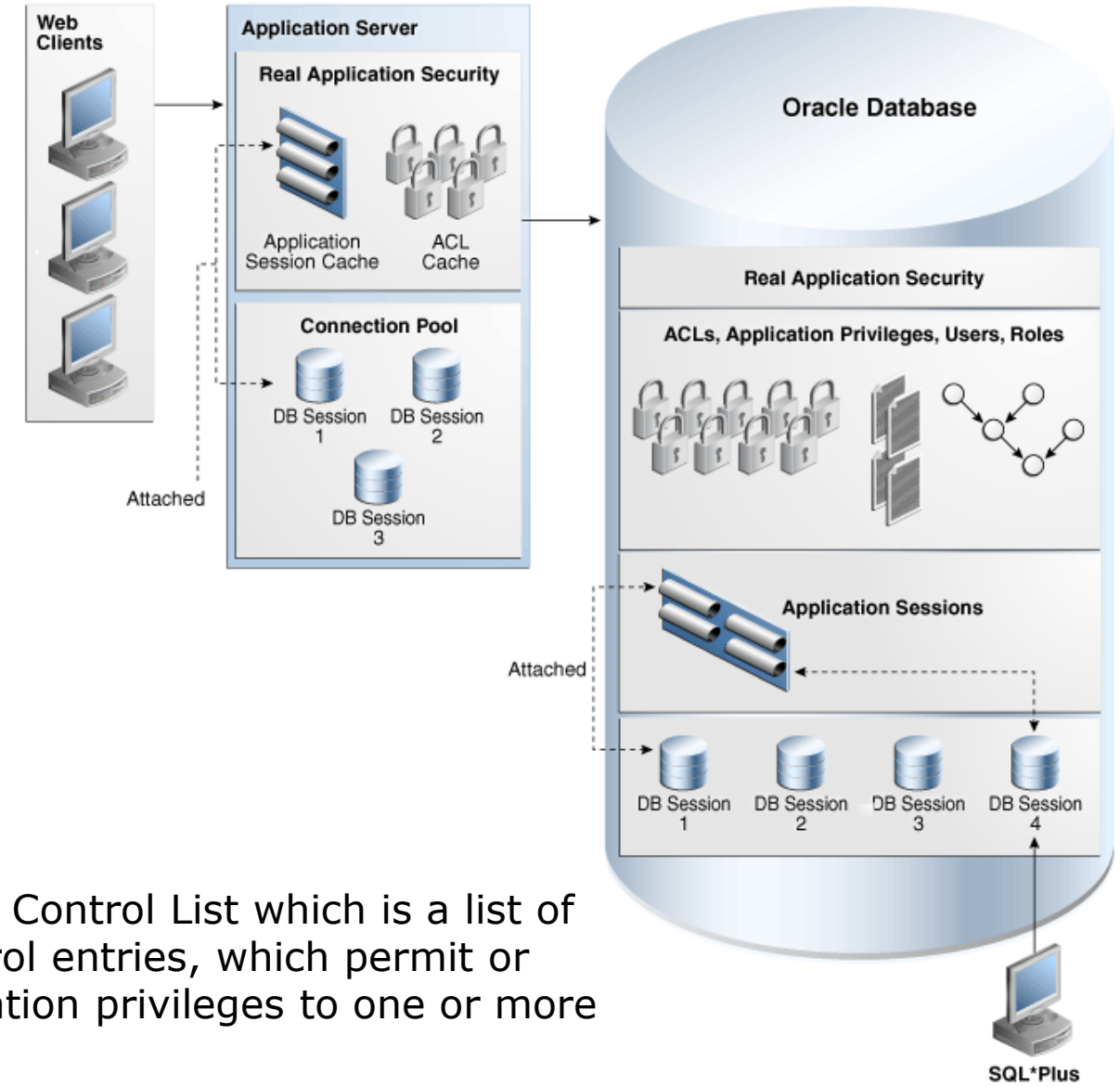
- Database access control: restricting data access and database activities, e.g. restrict users from querying specified tables or executing specified database statements.
- Privilege: the right to perform some action, e.g. run specific SQL statements.
- User: an account through which you can log in to the database, and to establish the means by which *Oracle Database (DBMS)* permits access by the user
- Role: a *named* set of privileges that can be granted to users or to other roles

Data Control Language - Access Control

- Privilege categories
  - System privilege
    - This is the right to perform a specific action in the database, or perform an action on any objects of a specific type. For example, CREATE USER and CREATE SESSION are system privileges.
  - Object privilege
    - This is the right to perform a specified action on an object, for example, query the employees table. Privilege types are defined by the database.

Source: Oracle 19c - Real Application Security Administrator's and Developer's Guide

Oracle Database Real Application Security Components – Oracle 19c



ACL: Access Control List which is a list of access control entries, which permit or deny application privileges to one or more principals.

GRANT Statement

- The GRANT statement provides privileges for a user to perform specific actions.

Create user **psmith** as a new user while granting **psmith** the **CREATE SESSION** system privilege.
If a password is specified using the **IDENTIFIED BY** clause, and the user name does not exist in the database, then a new user with that user name and password is created.
GRANT CREATE SESSION TO psmith IDENTIFIED BY *password*;

Grant the **UPDATE**, **INSERT**, and **DELETE** object privileges for all columns of the **EMPLOYEE** table to the user **psmith**.
GRANT UPDATE, INSERT, DELETE ON EMPLOYEE TO psmith;

The **WITH GRANT OPTION** clause with the **GRANT** statement can enable a grantee, psmith, to grant object privileges to *other* users.
GRANT SELECT ON EMPLOYEE TO psmith WITH GRANT OPTION;

Source: Oracle 19c – Oracle Database Security Guide

REVOKE Statement

- The REVOKE SQL statement revokes system privileges and roles.
- Any user with the ADMIN option for a system privilege or role can revoke the privilege or role from any other database user or role.
- The revoker does not have to be the user that originally granted the privilege or role. Users with GRANT ANY ROLE can revoke *any* role.
  - Can only revoke the privileges that the person who granted the privilege, directly authorized.
  - Cannot revoke grants that were made by other users to whom was granted the GRANT OPTION.
  - However, there is a cascading effect. If the object privileges of the user who granted the privilege are revoked, then the object privilege grants that were propagated using the GRANT OPTION are revoked.

Revoke **SELECT** and **INSERT** privileges on the **EMPLOYEE** table from user **psmith**.
REVOKE SELECT, INSERT ON EMPLOYEE FROM psmith;

Revoke all object privileges for **DEPARTMENT** table that was originally granted to the **human_resources** role.
REVOKE ALL ON DEPARTMENT FROM human_resources;

Source: Oracle 19c – Oracle Database Security Guide

GRANT and REVOKE Statements

```
-- from user 'root'@'localhost'
CREATE USER 'chau2'@'localhost' IDENTIFIED BY 'chau2';

GRANT ALL PRIVILEGES ON company.* TO 'chau2'@'localhost';

GRANT CREATE, ALTER, SELECT ON test.* TO 'chau2'@'localhost';

SELECT * FROM 'demo_20201003'.'employee';
```

Error Code: 1142. SELECT command denied to user 'chau2'@'localhost' for table 'employee'

```
GRANT SELECT ON demo_20201003.* TO 'chau2'@'localhost';

SHOW GRANTS FOR 'chau2'@'localhost';

REVOKE ALL PRIVILEGES ON *.* FROM 'chau2'@'localhost';
```

Source: a demonstration on MySQL

Data Control Language – Transaction Control

- Transaction: a logical data processing unit on a database
  - Change a database from a consistent state to another consistent one
  - ACID properties: atomic, consistent, isolated, durable
- Transaction Control Language: a set of commands which are used to mark the synchronization points (*sync points*) of transactions
  - COMMIT
  - ROLLBACK
  - SET AUTOCOMMIT OFF

Source: S. Chittayasothorn, *Relational Database Systems: Language, Conceptual Modeling and Design for Engineers*, Nutcha Printing Co. Ltd, 2017.

Data Control Language – Transaction Control

- COMMIT: confirms the operations in the transaction have successfully completed. It is the responsibility of the DBMS to make sure that all those confirmed operations are permanent.
- ROLLBACK: cancels all the operations from the previous sync point to the ROLLBACK point.
- SET AUTOCOMMIT OFF: equivalent to "begin transaction", opposite to SET AUTOCOMMIT ON. When AUTOCOMMIT is ON, each SQL command is considered as a separate transaction and is committed automatically after its execution. To group several SQL commands into one transaction, AUTOCOMMIT must be OFF.

Source: S. Chittayasothorn, *Relational Database Systems: Language, Conceptual Modeling and Design for Engineers*, Nutcha Printing Co. Ltd, 2017.

Database Stored Procedures and SQL/PSM

- Database program modules—procedures or functions—that are *stored and executed by the DBMS at the database server*.
  - These are historically known as *database stored procedures*, although they can be functions or procedures.
  - The term used in the SQL standard for stored procedures is *persistent stored modules* because these programs are stored persistently by the DBMS, similarly to the persistent data stored by the DBMS.

Database Stored Procedures and SQL/PSM

- Database stored procedures are useful in the following circumstances:
  - If a database program is needed by *several applications*, it can be stored at the server and invoked by any of the application programs. This reduces duplication of effort and improves software modularity.
  - Executing a program at the server can *reduce data transfer and communication cost* between the client and server in certain situations.
  - These procedures can enhance the modeling power provided by views by allowing *more complex types of derived data* to be made available to the database users via the stored procedures. Additionally, they can be used to check for *complex constraints* that are beyond the specification power of assertions and triggers.

Database Stored Procedures and SQL/PSM

- CREATE, ALTER, and DROP statements create, modify, and drop a stored procedure or a stored function.
- Functions/procedures can be written in SQL itself, or in an external programming language (e.g., C, Java).
  - Loops, if-then-else, assignment
- Some database systems (e.g., Oracle, MS SQL Server) support table-valued functions, which can return a table as a result.
- Functions are parameterized views that generalize the regular notion of views by allowing *parameters*.

Source: A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts – 6th Edition*, McGraw-Hill, 2006.

Stored Functions

- Given a department name, return the number of employees who work for the corresponding department.

```
create function dept_count (dept_name varchar (15))
returns int
begin
        declare d_count int;
                select count(*) into d_count
                from employee join department on dno = dnumber
                where dname = dept_name;
        return d_count;
end
```

```
select *
from department
where dept_count (dname) > 10;
```

DNAME	DNUMBER	MGR_SSN	MGR_START_DATE
Administration	4	987654321	1995-01-01
Research	5	333445555	1988-05-22

Stored Functions

```
CREATE FUNCTION [dbo].[dept_count]
(
    @dept_name varchar(15)
)
RETURNS int
AS
BEGIN

    DECLARE @d_count int;

    SET @d_count = (SELECT COUNT(*)
                    FROM dbo.employee join dbo.department on dno = dnumber
                    WHERE dname = @dept_name);

    -- Return the result of the function
    RETURN @d_count;

END
```

Source: a demonstration on MS SQL Server

Stored Functions

- Return the department number, department name, and average salary of each department of more than *n* employees.

```
create function dept_avg_salary (n int)
returns d_a_s TABLE (
                    dname           varchar (15),
                    dnumber         int,
                    avg_sal         decimal (10,2)
          )
begin
          insert into d_a_s
          select dname, dnumber, AVG(salary)
          from employee join department on dno = dnumber
          where dept_count(dname) > n
          group by dnumber, dname;
          return;
end
```

```
select *
from dept_avg_salary (2);
```

dname	dnumber	avg_sal
Administration	4	31000.00
Research	5	33250.00

Stored Functions

```
CREATE FUNCTION [dbo].[dept_avg_salary]
(
    @empnumber int
)
RETURNS
@d_a_s TABLE
(
    dname varchar(15),
    dnumber int,
    avg_sal decimal(10,2)
)
AS
BEGIN
    insert into @d_a_s
    select dname, dnumber, AVG(salary)
    from dbo.employee join dbo.department on dno = dnumber
    where dbo.dept_count(dname) > @empnumber
    group by dnumber, dname;
    RETURN
END
```

Source: a demonstration on MS SQL Server

Stored Procedures

- Given a department name, return the number of employees who work for the corresponding department.

```
CREATE PROCEDURE [dbo].[p_dept_count]
    (@dept_name varchar (15), @d_count int OUT)
AS
BEGIN
    SET @d_count = (select COUNT(*)
                    from employee join department on dno = dnumber
                    where dname = @dept_name);
END
```

```
declare @d_count int;
exec dbo.p_dept_count 'Research', @d_count output;
print @d_count;
```

4

Source: a demonstration on MS SQL Server

Stored Procedures

- Return the department number, department name, and average salary of each department of more than *n* employees.

```
CREATE PROCEDURE [dbo].[p_dept_avg_salary](@n INT)
AS
BEGIN
    select dname, dnumber, AVG(salary)
    from dbo.employee join dbo.department on dno = dnumber
    where dbo.dept_count(dname) > @n
    group by dnumber, dname;
END
```

```
DECLARE @dept_avg_sal TABLE (dname varchar(15), dnumber int, avg_salary decimal(10,2));
INSERT INTO @dept_avg_sal EXEC dbo.p_dept_avg_salary 2;
SELECT * FROM @dept_avg_sal;
```

dname	dnumber	avg_sal
Administration	4	31000.00
Research	5	33250.00

Source: a demonstration on MS SQL Server

Procedural language constructs

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
  - Warning: most database management systems implement their own variant of the standard syntax.
- Compound statement: **begin ... end**,
  - Multiple SQL statements are given between **begin** and **end**.
  - Local variables can be declared within a compound statements.
- **for**, **while** and **repeat** statements
- **if**, **if ... else ...** statements
- **case** statements, …

Source: A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts – 6th Edition*, McGraw-Hill, 2006.

Procedural language constructs

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

```
DECLARE cursor_name CURSOR FOR select_statement
```

```
OPEN cursor_name
```

```
FETCH cursor_name INTO var_name [, var_name] ...
```

```
CLOSE cursor_name
```

```
SET var_name = expr [, var_name = expr] ...
```

```
SELECT col_name [, col_name] ...
    INTO var_name [, var_name] ...
    table_expr
```

Source: a demonstration on MySQL

Procedural language constructs

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

```
LEAVE label
```

```
ITERATE label
```

```
RETURN expr
```

Source: a demonstration on MySQL

Stored Procedures

- Insert data of a new employee using 20000 if salary is <=0.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_employee`
    (id varchar(9), fname varchar(15), lname varchar(15), dno int(11), salary decimal(10,2))
BEGIN

    if salary <= 0.0 then
        set salary = 20000;
    end if;

    insert into employee (ssn, fname, lname, salary, dno)
    values (id, fname, lname, salary, dno);
END
```

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
fn_a	NULL	ln_b	135792468	NULL	NULL	NULL	20000.00	NULL	1
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Richard	K	Marini	653298653	1962-12-30	98 Oak Forest, Katy, TX	M	37000.00	653298653	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

```
call insert_employee ('135792468', 'fn_a', 'ln_b', 1, 0);
```

Source: a demonstration on MySQL

Stored Procedures

- Insert data of a new employee using the average salary of the department if no salary is specified (<=0).

```
CREATE PROCEDURE [dbo].[insert_employee]
    (@id varchar(9), @fname varchar(15), @lname varchar(15), @dno int, @salary decimal(10,2))
AS
BEGIN

    declare @avg_sal decimal(10,2)

    if (@salary<=0)
        SET @avg_sal = (select avg(salary)
                        from employee
                        where dno = @dno)
    else
        SET @avg_sal = @salary

    insert into employee (ssn, fname, lname, salary, dno)
    values (@id, @fname, @lname, @avg_sal, @dno)

END
```

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPER_SSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
fn_a	NULL	ln_b	135792468	NULL	NULL	NULL	31000.00	NULL	4
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	987654321	4

```
EXEC dbo.insert_employee '135792468', 'fn_a', 'ln_b', 4, 0;
```

Source: a demonstration on MS SQL Server

Triggers

- A trigger is a named database object that is associated with a table, and that *automatically* activates when a particular event occurs for the table.
  - Activation time
    - BEFORE or AFTER an event
    - *INSTEAD OF* an event
  - Event: INSERT, DELETE, UPDATE
  - Refer to columns in the table associated with the trigger by using the aliases OLD (***deleted***) and NEW (***inserted***).
    - OLD.col_name: a column of an existing row before it is updated or deleted
    - NEW.col_name: a column of a new row to be inserted or an existing row after it is updated

Triggers

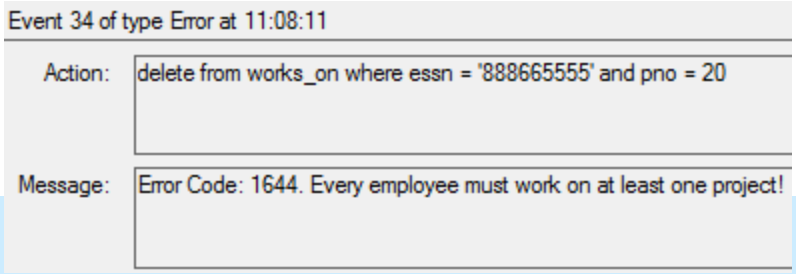
- When to use triggers
  - Enforce application-based semantic constraints
  - Maintain summary data
  - Replicate databases by recording changes to other special tables
- When not to use triggers **Cascading execution**
  - Encapsulation facilities available for update (insert, delete, update) methods with constraints
  - Materialized views available for maintaining summary data
  - Built-in DBMS support ready to replicate databases

Source: A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts – 6th Edition*, McGraw-Hill, 2006.

Triggers

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_body
```

Create a trigger to ensure each employee works on at least one project.




```
DELIMITER $$

CREATE TRIGGER trgTotalEmployee_after_delete_works_on AFTER DELETE ON works_on
FOR EACH ROW
BEGIN

    declare total_participation condition for SQLSTATE '45000';

    IF NOT EXISTS (select * from works_on where essn = old.essn) THEN
        SIGNAL total_participation
            SET MESSAGE_TEXT = 'Every employee must work on at least one project!';
    END IF;

END $$
```

Source: a demonstration on MySQL

Triggers

- Update the number of employees of the department from which an employee is removed.

```
CREATE TRIGGER delete_trigger ON employee AFTER DELETE
AS
BEGIN
    DECLARE @dname VARCHAR(15);
    DECLARE @count INT;

    SELECT @dname = dname, @count = COUNT(*)
    FROM department join employee ON dnumber = dno
    WHERE dno IN (SELECT distinct dno FROM deleted)
    GROUP BY dname;

    PRINT 'Department name: ' + @dname + ' -- Number of employees: ' + CAST(@count AS VARCHAR(15));
END
```

Source: a demonstration on MS SQL Server

Triggers

- Check if a new employee is with the same department as his/her supervisor before inserted.

```
CREATE TRIGGER insert_emp_trigger ON employee INSTEAD OF INSERT
AS
BEGIN
    IF (EXISTS(SELECT * FROM employee WHERE ssn = (select super_ssn from inserted)
                                        AND dno = (select dno from inserted)))
    BEGIN
        INSERT INTO employee
        SELECT * FROM inserted;
    END
END
```

Source: a demonstration on MS SQL Server

NOTES on Functions, Procedures, and Triggers

- More details of stored functions, stored procedures, and triggers
  - Check the specification of a particular DBMS
  - See more examples on MS SQL Server with a demonstration uploaded in BKEL

Summary

- SQL = Structured Query Language
  - Data Definition Language
  - Data Manipulation Language
  - Data Control Language
  - Stored Functions
  - Stored Procedures
  - Triggers

ANSI/ISO Standard: SQL-86, SQL-89, SQL-92, SQL-1999, SQL-2003, SQL-2006, SQL-2008, SQL-2011, SQL-2016, *SQL-2019*

→ A *declarative* non-procedural language, based on tuple relational calculus, supported by existing RDBMSs

→ Associated with relational algebra for *internal* representation, processing, and optimization

Summary

- Further readings
  - DDL statements for views, indexes
  - Full syntax of each SQL statement on a specific relational DBMS
    - Oracle
    - MS SQL Server
    - MySQL
    - PostgreSQL
    - SQLite
    - …

Chapter 4: The SQL Language

Review

- 1. Given the Company database, write a SELECT statement for each requirement:
  - 1.1. Retrieve Ssn, name, and address of the employees who work in department 5.
  - 1.2. Retrieve Ssn, name, and address of the employees who work in 'Research' department.
  - 1.3. Retrieve Ssn, name, and address of the employees who work in departments in Houston.
  - 1.4. Return the average salary of the employees who work in department 1.
  - 1.5. Return the maximum salary of the employees in each department.
  - 1.6. Return the number of the employees who have salaries greater than the average salary of the employees who work in department 1.

Review

- 1. Given the Company database, write a SELECT statement for each requirement:
  - 1.7. 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.
  - 1.8. Find the names of employees who work on all the projects controlled by department number 5.
  - 1.9. Make a list of project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.
  - 1.10. List the names of all employees with two or more dependents in department 5.
  - 1.11. Retrieve the names of employees who have no dependents.

Review

- 1. Given the Company database, write a SELECT statement for each requirement:
  - 1.12. List the names of managers who have at least one dependent.
  - 1.13. Retrieve the names of all employees in department 5 who work more than 10 hours per week on the ProductX project.
  - 1.14. List the names of all employees who have a dependent with the same first name as themselves.
  - 1.15. Find the names of all employees who are directly supervised by 'Franklin Wong'.
  - 1.16. For each project, list the project name and the total hours per week (by all employees) spent on that project.
  - 1.17. Retrieve the names of all employees who work on every project.

Review

- 1. Given the Company database, write a SELECT statement for each requirement:
  - 1.18. Retrieve the names of all employees who do not work on any project.
  - 1.19. For each department, retrieve the department name and the average salary of all employees working in that department.
  - 1.20. Retrieve the average salary of all female employees.
  - 1.21. Find the names and addresses of all employees who work on at least one project located in Houston but whose department has no location in Houston.
  - 1.22. List the last names of all department managers who have no dependents.
  - 1.23. List the department name, the number of projects, and the number of employees of each department that controls those projects in the same location.

Review

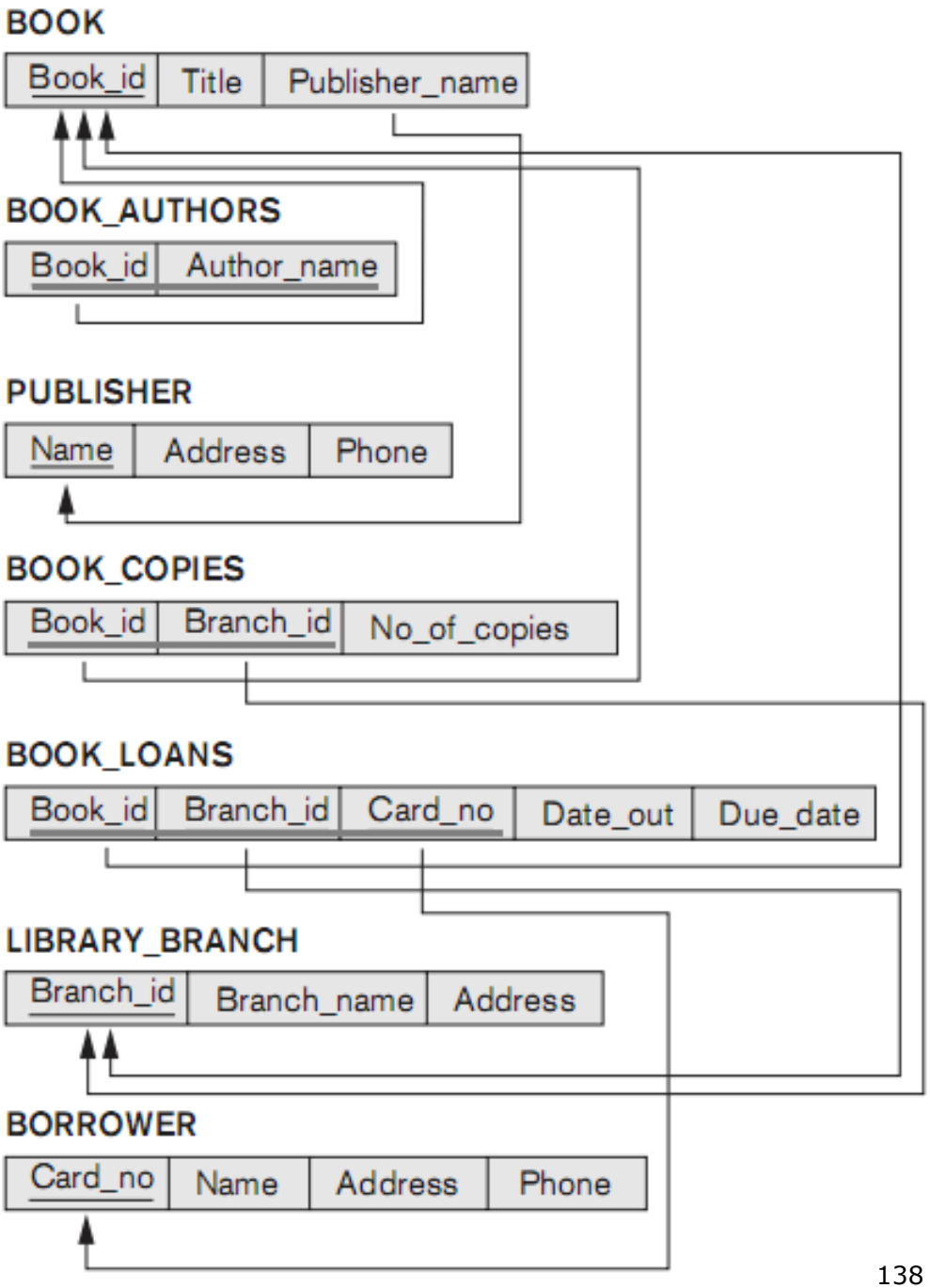
- 2. For the Company database, write a DML statement for each requirement. Is any constraint violated?
  - 2.1. Insert <'Robert', 'F', 'Scott', '943775543', '1972-06-21', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1> into EMPLOYEE.
  - 2.2. Insert <'ProductA', 4, 'Bellaire', 2> into PROJECT.
  - 2.3. Insert <'Production', 4, '943775543', '2007-10-01'> into DEPARTMENT.
  - 2.4. Insert <'677678989', NULL, '40.0'> into WORKS_ON.
  - 2.5. Insert <'453453453', 'John', 'M', '1990-12-12', 'spouse'> into DEPENDENT.
  - 2.6. Delete the WORKS_ON tuples with Essn = '333445555'.
  - 2.7. Delete the EMPLOYEE tuple with Ssn = '987654321'.
  - 2.8. Delete the PROJECT tuple with Pname = 'ProductX'.
  - 2.9. Delete the DEPARTMENT tuple with Dname = 'Administration'.

Review

- 2. For the Company database, write a DML statement for each requirement. Is any constraint violated?
  - 2.10. Modify the Mgr_ssn and Mgr_start_date of the DEPARTMENT tuple with Dnumber = 5 to '123456789' and '2007-10-01', respectively.
  - 2.11. Modify the Super_ssn attribute of the EMPLOYEE tuple with Ssn = '999887777' to '943775543'.
  - 2.12. Modify the Hours attribute of the WORKS_ON tuple with Essn = '999887777' and Pno = 10 to '5.0'.
  - 2.13. Modify the Dname attribute of the DEPARTMENT tuple with Dname = 'Administration' to 'Headquarters'.
  - 2.14. Modify the Salary attribute of the EMPLOYEE tuples with the average salary of the department 'Research' where the salary of its manager is 10% higher.
  - 2.15. Modify the Dno attribute of the EMPLOYEE tuples with Dno = 1 to Dno = 2.

Review

- 3. Given the relational schema of the LIBRARY database, write DDL statements to implement this schema with any assumption needed for data types and constraints.



Review

- 4. Write SQL statements on the LIBRARY database in question 3.
  - 4.1. Add an ISBN attribute of varchar(13) into BOOK relation schema where ISBN is a secondary key.
  - 4.2. How many copies of the book titled The Lost Tribe are owned by the library branch whose name is 'Sharpstown'?
  - 4.3. How many copies of the book titled The Lost Tribe are owned by each library branch?
  - 4.4. Retrieve the names of all borrowers who do not have any books checked out.
  - 4.5. For each book that is loaned out from the Sharpstown branch and whose Due_date is today, retrieve the book title, the borrower's name, and the borrower's address.
  - 4.6. For each library branch, retrieve the branch name and the total number of books loaned out from that branch.
  - 4.7. Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.
  - 4.8. For each book authored (or coauthored) by Stephen King, retrieve the title and the number of copies owned by the library branch whose name is Central.

Review

- 5. Write stored functions to process the data in the LIBRARY database as follows:
  - 5.1. Return ISBN and book title of each book available in every library branch.
  - 5.2. Given a publisher name and a date, return the number of books published by this publisher borrowed on the date.
  - 5.3. Given an author name, return the details of the borrowers who borrowed his/her books the most.

Review

- 6. Write stored procedures to process the data in the LIBRARY database as follows:
  - 6.1. Given a date, update a temporary table with the details of all the books that are not available on that date because all of them were borrowed.
  - 6.2. Given an author name, return the most recent date when his/her book was borrowed.
  - 6.3. Return the library branches where the books have been borrowed the most.

Review

- 7. Write triggers to process the data in the LIBRARY database as follows:
  - 7.1. Enforce the constraint which is: the number of copies of each book in any library branch must be at least 2.
  - 7.2. Enforce the constraint which is: a borrower cannot borrow more than 3 books at the same time.
  - 7.3. Update a temporary table of the books whose copies are not available in any library branch because all of them were borrowed.

Next
Chapter 5: Relational Database Design

- 5.1. Design Guidelines for Relation Schemas
- 5.2. Functional Dependencies
- 5.3. Normal Forms Based on Primary Keys
- 5.4. Boyce-Codd Normal Form
- 5.5. Properties of Relational Decompositions
- 5.6. Algorithms for Relational Database Schema Design