CS3402 – Chapter 5
Integrity Constraints

Integrity Constraints

- Constraints determine which values are permissible and which are not in the database (table)
  - Constraints are conditions that must hold on all valid relation states

- A relational database schema S is a set of relation scheme S = {R1, R2, …, Rn} and a set of integrity constraints IC

- Valid state Vs. invalid state
  - Invalid state: A database state that does not obey all the integrity constraints
  - Valid state: a state that satisfies all the constraints in the defined set of integrity constraints

Relational Integrity Constraints

- They are of three main types of constraints:
  - Inherent or Implicit Constraints: These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
  - Schema-based or Explicit Constraints: They are expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
  - Application-based or Semantic constraints: These are beyond the expressive power of the model and must be specified and enforced by the application programs

Relational Integrity Constraints

- There are three *main types* of schema-based constraints that can be expressed in the relational model:
    - Key constraints
    - Entity integrity constraints
    - Domain constraint
    - Referential integrity constraints

Key Constraints

- Superkey of R:
  - A set of attributes that contains a key is called a superkey
  - It is a set of attributes SK, e.g., {A1, A2} of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - For any distinct tuples t1 and t2 in r(R), t1[SK] $\neq$ t2[SK]

- Key (Primary key, Candidate Key) of R:
  - A "minimal" superkey
  - A key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Key Constraints

- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys:
    - Key1 = {State, Reg#}
    - Key2 = {SerialNo}
  - Both are also superkeys of CAR
  - {SerialNo, Make} is a superkey but *not* a key

- In general:
  - Any *key* is a *superkey* (but not vice versa)
  - Any set of attributes that *includes a key* is a *superkey*
  - A *minimal* superkey is also a key

Key Constraints

- If a relation has several candidate keys, one is chosen arbitrarily to be the primary key
  - The primary key attributes are <u>underlined</u>
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, <u>SerialNo</u>, Make, Model, Year)
  - We chose SerialNo as the primary key
- The primary key value is used to *uniquely identify* each tuple in a relation
- General rule: Choose as primary key the smallest of the candidate keys (in terms of size)

Key Constraints

- **Key** constraints

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 5.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

Keys of Relations

- A set of one or more attributes {A1, A2, …, An} is key for a relation if:
  - The attributes functionally determine all other attributes of the relation
  - Relations are sets. It is impossible for two distinct tuples of R to agree on all A1, A2, …, An
  - No proper subset of {A1, A2, …, An} functionally determines all other attributes of R, i.e., a key must be minimal

Entity Integrity Constraints

- Entity Integrity:
  - The *primary key attributes* PK of each relation schema R cannot have null values in any tuple of R
    - Primary key values are used to *identify* the individual tuples
    - $t[PK] \neq$ null for any tuple t in R
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key

Domain Constraints

- *Domain constraint:* Every value in a tuple must be from the *domain of its attribute* (or it could be null, if allowed for that attribute)

E.g.,

C-Name: string of char (30)

Balance: Number (6,2)

…

Referential Integrity

- Key and entity integrity constraints are specified on individual relations
- Referential integrity is a constraint involving two relations
  - To specify a relationship among tuples in two relations
  - The referencing relation and the referenced relation (R1 -> R2)
- Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2 if it satisfies:
  - The attributes in FK have the same domain(s) as the primary key attributes PK of R2

Referential Integrity

- Referential integrity constraints typically arise from the relationships among the entities represented by the relation
- For example, in the EMPLOYEE relation, the attribute Dno refers to DEPARTMENT for which an employee works
- We designate Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT
- A value of Dno in any tuple t1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT, Dnumber, in the same tuple t2 of the DEPARTMENT relation
- Or the value of Dno can be NULL if the employee does not belong to a department or will be assigned to a department later

Integrity Constraints

- **Referential Integrity Constraints**
  - *typically this implies some "subset dependency" relationships between 2 sets of attributes in 2 tables*

    E.g.,

Supplier

S#	Sname …..
S1	
S2	
S3	
...	

Shipment

Sno	Pno	Qty
S1	P1	15
S1	P2	25
S2	P2	20

ie, Supplier[S#] $\supseteq$ *Shipment[Sno]*

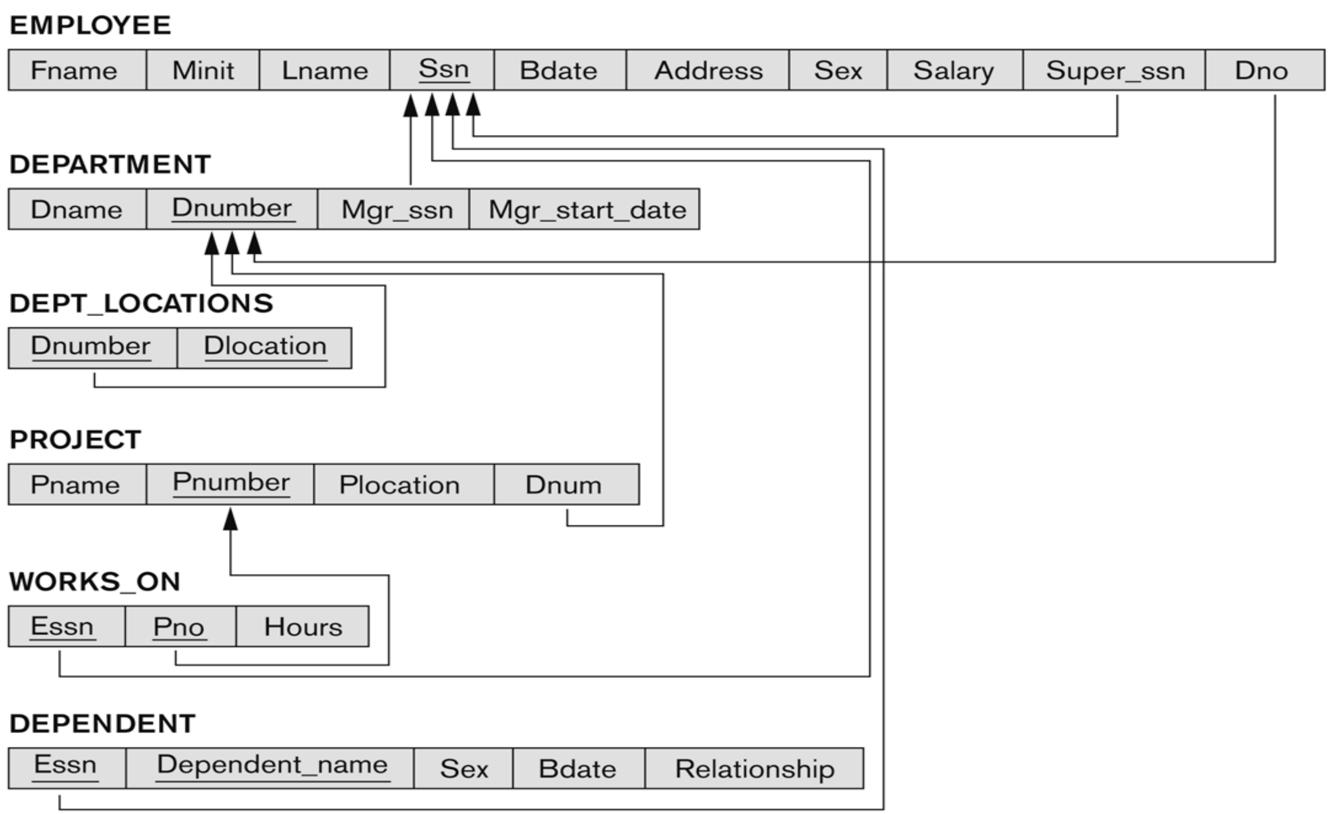
Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
- Next slide shows the COMPANY relational schema diagram with referential integrity constraints

Database State for COMPANY

- All examples discussed below refer to the COMPANY database shown here

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Populated database state for COMPANY

Figure 5.6
One possible database state for the COMPANY relational database schema.

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

SQL CREATE TABLE data definition statements for defining the COMPANY schema

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

SQL CREATE TABLE data definition statements for defining the COMPANY schema

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

Update Operations on Relations

- INSERT a tuple
- DELETE a tuple
- MODIFY a tuple

- Integrity constraints should not be violated by the update operations
- Several update operations may have to be grouped together
- Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints

Possible violations for each operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database

- INSERT may violate any of the constraints:
  - Domain constraint: if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint: if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity: if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity: if the primary key value is null in the new tuple

Integrity Constraints

- In case of integrity violation, several actions can be taken:
  - cancel the operation that causes the violation
  - perform the operation but inform the user of the violation
  - trigger additional updates so the violation is corrected
  - execute a user-specified error-correction routine

Adding Constraints in SQL

- ```
  CREATE TABLE TOY
      (toy_id  NUMBER(10),
       description VARCHAR(15)      NOT NULL,
       last_purchase_date DATE,
       remaining_qnt NUMBER(6));
  ```

- ```
  CREATE TABLE TAB1
      (col1 NUMBER(10)              PRIMARY KEY,
       col2 NUMBER(4)               NOT NULL,
       col3 VARCHAR(5)              REFERENCES zipcode(zip)
                                    ON DELETE CASCADE,
       col4 DATE,
       col5 VARCHAR(20)             UNIQUE,
       col6 NUMBER(5)               CHECK (col6 < 100));
  ```

Naming Constraints in SQL

```
CREATE TABLE TAB1
      (col1 NUMBER(10)           PRIMARY KEY,
       col2 NUMBER(4)            NOT NULL,
       col3 VARCHAR(5)           REFERENCES zipcode(zip)
                                 ON DELETE CASCADE,
       col4 DATE,
       col5 VARCHAR(20)          UNIQUE,
       col6 NUMBER(5)            CHECK (col6 < 100)),
       CONSTRAINT TAB1_PK        PRIMARY KEY(col1)
       CONSTRAINT TAB1_ZIPCODE_FK FOREIGN KEY(col3)
                                 REFERENCES ZIPCODE(ZIP)
       CONSTRAINT TAB1_COL5_UK UNQIUE(col5),
       CONSTRAINT TAB1_COL6_CK CHECK (col6 < 100);
```

Reference Constraints in SQL

- CREATE TABLE COUNTRY

```
(cntry_cd    CHAR(3)                  NOT NULL,
 cname       VARCHAR2(32)             NOT NULL,
 ename       VARCHAR2(32)             NOT NULL,
 curr_cd     CHAR(3)                  NOT NULL,
 upd_dt      DATE DEFAULT SYSDATE     NOT NULL ,
 upd_uid     VARCHAR2(16)             NOT NULL);
```

- CREATE TABLE EXCHANGE

```
(exchg_cd    VARCHAR2(8)              NOT NULL,
 cname       VARCHAR2(32)             NOT NULL,
 ename       VARCHAR2(32)             NOT NULL,
 cntry_cd    CHAR(3)                  NOT NULL);
```

Reference Constraints in SQL

- CREATE TABLE COUNTRY

ALTER TABLE COUNTRY ADD CONSTRAINT PK_country PRIMARY KEY (cntry_cd);

ALTER TABLE EXCHANGE ADD CONSTRAINT PK_exchange PRIMARY KEY (exchg_cd);

ALTER TABLE EXCHANGE ADD CONSTRAINT FK_exchg_cntry FOREIGN KEY (cntry_cd) REFERENCES COUNTRY (cntry_cd) ;

Functional Dependency

- Functional dependency is a constraint between two sets of attributes from the database
- E.g., In DEPARTMENT, *deptno* and *dname*
  - If you know the department number, you know the department name
- A functional dependency denotes by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of a relation R specifies a constraint on the possible tuples that can form a relation state r of R
- The constraint is that, for any two tuples t1 and t2 in r that have $t1[X] = t2[X]$, they must also have $t1[Y] = t2[Y]$
- The values of the Y component of a tuple in r depend on, or are determined by the values of the X component
- If you know his student ID, then I know his name $(X \rightarrow Y)$

Functional Dependency

- Formal definition:
  - Let R be a relation schema, and $\alpha \subseteq R$, $\beta \subseteq R$ (i.e., α and β are sets of R's attributes). We say:

$$\alpha \rightarrow \beta$$

  - If in any relation instance r(R), for all pairs of tuples t1 and t2 in r, we have:

$$(t1[\alpha] = t2[\alpha]) \Rightarrow (t1[\beta] = t2[\beta])$$

- E.g., for the table Borrow (B-name, Loan#, C-name, Amount),

  we have: Loan# $\rightarrow$ Amount

  (read as "uniquely determines")

Functional Dependency

- Some usages of FDs:
  (1) to set constraints on legal relations. (e.g. Key constrains)
  Eg, Loan# $\rightarrow$ Amount (as in the Borrow table example)

  (2) to test relations to see if they are "legal" under a given set of FDs.

  (3) to be used in designing the database schema.

Functional Dependency: keys

- Candidate key
  - If a constraint on R states X is a candidate key of R, then X->Y for any subset of attributes Y of R
  - A candidate key uniquely identifies a tuple
  - The values of all remaining attributes are determined

- A functional dependency is property of the semantics or meaning of the attributes

Functional Dependency: Keys

Movies(title, year, length, type, studioName, starName)
Title, year, starName -> length, type, studioName

- Attributes {title, year, starName} form a key for the relation Movie
- Suppose two tuples agrees on these three attributes: title, year, starName
- They must agree on the other attributes, length, type and studioName
- No proper subset of {title, year, starName} functionally determines all other attributes
- {title, year} does not determine starName since many movies have more than one star
- {year, starName} is not a key because we could have a star in two movies in the same year

Functional Dependency: properties

- E.g., for the table Borrow (B-name, Loan#, C-name, Amount)

- If X → Y in R, this does not say whether or not Y → X in R
  - Amount → Loan#? No
- If X = Loan#; Y = Amount and Loan# → Amount
  - Can it be {Loan#, B-name} → Amount? Yes

- Some FDs are "trivial", since they are always satisfied by all relations:
  - E.g., A → A, AB → A,
  - E.g., {C-name, Amount} → C-name

- An FD is trivial if and only if the right-hand side (the dependent) is a subset of the left-hand side (the determinant), e.g., AB → A

Inference Rules for FDs

- Given a set of FDs F, we can infer additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules:
  - IR1. (Reflexive) If Y is a *subset of* X, then X → Y
  - IR2. (Augmentation) If X → Y, then XZ → YZ
    - ✓ (Notation: XZ stands for X U Z)
  - IR3. (Transitive) If X → Y and Y → Z, then X → Z
- IR1, IR2, IR3 form a sound and complete set of inference rules
  - Sound: These rules are true
  - Complete: All the other rules that are true can be deduced from these rules

Inference Rules for FDs

- Some additional inference rules that are useful:
  - Decomposition**:** If X → YZ, then X → Y and X → Z
  - Union: If X → Y and X → Z, then X → YZ
  - Pseudotransitivity**:** If X → Y and WY → Z, then WX → Z

Inference Rules for FDs

IR1 (reflective rule)	If X is a subset of Y, then X → Y
IR2 (augmentation rule)	If X → Y, then XZ → YZ
IR3 (transitive rule)	If X → Y and Y → Z, then X → Z
IR4 (decomposition rule)	If X →YZ, then X → Y and X → Z
IR5 (union rule)	If X → Y and X → Z, then X → YZ
IR6 (pseudotransitive rule)	If X → Y and WY → Z, then WX → Z

Example

- Suppose we are given a schema R with attributes A, B, C, D, E, F and the FDs are:
  - $A \rightarrow BC$
  - $B \rightarrow E$
  - $CD \rightarrow EF$
  - Show that the FD $AD \rightarrow F$ holds

1. $A \rightarrow BC$ (given)
2. $A \rightarrow C$ (1, decomposition)
3. $AD \rightarrow CD$ (2, augmentation)
4. $CD \rightarrow EF$ (given)
5. $AD \rightarrow EF$ (3 and 4, transitivity)
6. $AD \rightarrow F$ (5, decomposition)

Inference Rules for FDs

- Closure of a set F of FDs is the set F^+ of all FDs that can be inferred from F
- E.g., suppose we specify the following set F of obvious functional dependencies
  - F = {Ssn →{Ename, Bdate, Address,Dnumber}, Dnumber→{Dname, Dmgr_ssn}}
  - Then,
    - Ssn → {Dname, Dmgr_ssn}
    - Ssn → Ssn
    - Dnumber → Dname

Inference Rules for FDs

- Closure of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
  - Note both X and X^+ are a set of attributes
- If X^+ consists of all attributes of R, X is a superkey for R
  - From the value of X, we can determine the values the whole tuple
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F
- From X to find out X^+

Example

- Suppose we are given a schema R with attributes A, B, C, D, E, F, and FDs
- $A \rightarrow BC$
- $E \rightarrow CF$
- $B \rightarrow E$
- $CD \rightarrow EF$
- $\{A\}^+ =>$

$\{A\}^+ = \{A,B,C,E,F\}$
Not a superkey or a key

Example

- F = {Ssn $\rightarrow$ Ename
  Pnumber $\rightarrow$ {Pname, Plocation}
  {Snn, Pnumber} $\rightarrow$ Hours}

- The following closure sets with respect to F
  - {Ssn}$^+$ = {Ssn, Ename}
  - {Pnumber}$^+$ = {Pnumber, Pname, Plocation}
  - {Ssn, Pnumber}$^+$ = {Ssn, Pnumber, Ename, Pname, Plocation, Hours}

Equivalence of Sets of FDs

- A set of functional dependencies F is said to cover another set of functional dependency E if every FD in E is also in F^+ (E is a subset of F^+)

- Two sets of FDs F and G are equivalent if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$

- Example:
  - F: $A \rightarrow BC$; {$A \rightarrow B$, $A \rightarrow C$ (decomposition rule)}
  - G: $A \rightarrow B$, $A \rightarrow C$
  - $F^+ = G^+$

Summary

- Closure of a set F of FDs
  - The set F^+ of all FDs that can be inferred from F
- Closure of a set of attributes X with respect to F
  - The set X^+ of all attributes that are functionally determined by X
- A set of functional dependencies F is said to cover another set of functional dependency E
  - If every FD in E is also in F^+
- Two sets of FDs F and G are equivalent
  - If F and G are equivalent if $F^+ = G^+$

References

- 6e
  - Ch. 3, p. 63 – 70